2009-12

A field programmable gate array based software defined radio design for the space environment

Livingston, Jeremy V.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A FIELD PROGRAMMABLE GATE ARRAY BASED SOFTWARE DEFINED RADIO DESIGN FOR THE SPACE ENVIRONMENT

by

Jeremy V. Livingston

December 2009

Thesis Advisor:	Frank E. Kragh
Co-Advisor:	Herschel Loomis

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2009	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE A Field Programmable Gate Array Based Software Defined Radio Design for the Space Environment		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Jeremy V. Livingston			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

This thesis focuses on a software defined radio (SDR) designed to compress a wideband radio signal input for a narrowband signal output. The design is based on a Field Programmable Gate Array (FPGA), which is chosen for its reprogrammability, flexibility, and our ability to introduce fault tolerance into the design. Software design tools allowed programming to be done at a high level, thereby allowing more progress on the design. This thesis focuses on one such SDR that was designed at a high level of abstraction. This thesis documents an analysis of the memory and timing requirements of the circuit so that it may be used on resource-constrained FPGA devices. It also explores the operating capabilities and limitations for this circuit under various resource-constrained conditions and introduces algorithms for fault detection to make the circuit more compatible with the space environment.

14. SUBJECT TERMS Data Compression, Signal Analysis, Software Defined Radio (SDR), System Generator, Fast Fourier Transform (FFT), Field Programmable Gate Array (FPGA), Xilinx, Virtex™, Error Detection, Parity, Space-Based Computing			**15. NUMBER OF PAGES** 129
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A FIELD PROGRAMMABLE GATE ARRAY BASED SOFTWARE DEFINED RADIO DESIGN FOR THE SPACE ENVIRONMENT

Jeremy V. Livingston
Lieutenant, United States Navy
B.S., Rensselaer Polytechnic Institute, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2009

Author: Jeremy V. Livingston

Approved by: Frank E. Kragh
Thesis Advisor

Herschel Loomis
Co-Advisor

Jeffrey Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis focuses on a Software Defined Radio (SDR) designed to compress a wideband radio signal input for a narrowband signal output. The design is based on a Field Programmable Gate Array (FPGA), which is chosen for its reprogrammability, flexibility, and our ability to introduce fault tolerance into the design. Software design tools allowed programming to be done at a high level, thereby allowing more progress on the design. This thesis focuses on one such SDR that was designed at a high level of abstraction. This thesis documents an analysis of the memory and timing requirements of the circuit so that it may be used on resource-constrained FPGA devices. It also explores the operating capabilities and limitations for this circuit under various resource-constrained conditions and introduces algorithms for fault detection to make the circuit more compatible with the space environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

BRAM	Block Random Access Memory
CLB	Configurable Logic Block
CFTP	Configurable Fault-Tolerant Processor
CRL	Communications Research Laboratory
DC	Direct Current
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
DTFT	Discrete Time Fourier Transform
FIFO	First-In, First-Out Memory
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FT	Fourier Transform
GCLK	Global Clock Buffer
GUI	Graphical User Interface
HDL	Hardware Description Language
IDFT	Inverse Discrete Fourier Transform
IDTFT	Inverse Discrete Time Fourier Transform
IF	Intermediate Frequency
IOB	Input/Output Block
IP	Intellectual Property
ISE	Integrated Synthesis Environment
LUT	Look-Up Table

NPS	Naval Postgraduate School
ORS	Operationally Responsive Space
RAM	Random Access Memory
RFD	Ready for Data
ROI	Range of Interest
RPR	Reduced Precision Redundancy
SDR	Software Defined Radio
SEU	Single Event Upset
TMR	Triple Modular Redundancy
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XOR	Exclusive Or

EXECUTIVE SUMMARY

The acquisition of satellite systems is a process that typically takes several years between the identification of a need to the delivery of a capability. Operationally Responsive Space (ORS) is a strategy that strives to deliver space-based assets to the war fighter in a timely manner. Two important areas of concern are improving the flexibility of satellite designs and streamlining the acquisition process. The use of space-based Field Programmable Gate Arrays (FPGAs) can leverage both of these key ideas. This thesis work is intended to demonstrate the feasibility of using a space-based FPGA system as an option for ORS. This is accomplished by implementing a tactically relevant Software Defined Radio (SDR) algorithm in a configuration suitable for the space environment.

A FPGA-based SDR design was developed through a previous thesis project conducted by a student working through the Naval Postgraduate School (NPS) Communications Research Laboratory (CRL). The circuit computes the Fast Fourier Transform (FFT) of a sampled real-time Intermediate Frequency (IF) signal. The complex FFT result is stored in temporary memory while the energy in the signal is analyzed. The circuit uses operator-defined time windows and frequency Ranges of Interest (ROI) to organize the FFT output into time-frequency bins. Each bin is compared with a user-defined minimum energy threshold. The circuit compresses the FFT output by discarding all time-frequency bins that do not meet the minimum threshold. Bins that pass the threshold analysis are retrieved from temporary memory and forwarded to the circuit's output. The circuit is also designed so that *N*, the number of points processed by the FFT, can be adjusted.

The initial SDR design was created and tested at a high level of abstraction using System Generator software produced by Xilinx. This software interfaces with the MATLAB®/Simulink® environment. System Generator provides a library of pre-designed Intellectual Property (IP) modules which perform functions within a Simulink® model. The Simulink® model can be used to generate files required to program a FPGA

with the design. Tests were conducted in the MATLAB®/Simulink® environment to verify functionality of the design, as documented in previous thesis work. The design was tested using a configuration with $N = 8$ on a Virtex™-4 FPGA.

The first goal of this research was to find ways to implement the design with a high value of *N* within the resource constraints of selected FPGA models. Although the initial SDR design functions in the MATLAB®/Simulink® environment, when *N* is increased from $N = 8$ to $N = 1024$, the design requires more memory resources than are available on a Virtex™-4 FPGA. The high level of abstraction used for the initial SDR design allowed the designer to bypass a detailed analysis of the circuit's timing and resource requirements. This led to several inefficiencies throughout the design that needed to be corrected for the circuit to function with a high value of *N*.

This thesis describes an analysis of the circuit elements produced using System Generator to better understand the function of the design. IP modules used to compute the FFT were examined by conducting a series of tests in the MATLAB®/Simulink® environment. The results of these tests verified that the expected output signals are generated from a series of different input signals. The Xilinx Integrated Synthesis Environment (ISE) Project Navigator software was used to check the FPGA resources required for these IP modules. Post-synthesis timing was verified for some tests using ModelSim® simulation software.

The FFT IP module used in the initial SDR design is the FFTv4.1 IP block. The data sheet for this IP circuit design explains which target FPGA devices can use it. Eligible target devices include devices in the Virtex™-4 and Virtex™-IIP family of FPGAs. The list of eligible devices excludes the Virtex™-I FPGA family, which is the target device for several legacy space-configured FPGA systems including the NPS Configurable Fault Tolerant Processor (CFTP) experiment. For this reason, a FFT IP module suitable for the Virtex™-I was also examined. A look at all IP modules in the System Generator library revealed that the FFTv1.0 IP block is the only one that meets this constraint.

The FFTv1.0 IP block functions differently from the FFTv4.1 IP block in several ways. The FFTv4.1 IP block samples the input signal every clock cycle, while the FFTv1.0 IP block samples the input signal once every four clock cycles. The FFTv4.1 IP block can be configured for full-rate pipeline operations so that after an initial latency, a valid output signal is produced on every clock cycle. In contrast, the FFTv1.0 IP block produces a single burst of N valid output signals every $4N$ clock cycles. Because the initial SDR is configured to accept the continuous streaming output of the FFTv4.1 IP block, the difference between these FFT IP circuits means that changes to the initial SDR design are required if a Virtex™-I FPGA is the desired target device.

In addition to analyzing the FFT IP modules, the control algorithms that govern signal flow through the circuit were examined. Simulations in the MATLAB®/Simulink® environment were used to create state transition diagrams explaining the behavior of the control algorithms. The results of this analysis were used to create expressions for the timing expectations in terms of the size of the FFT sample period N, the number of FFT periods in each time window M, and the sum of the sizes of all user-defined ROI. Expressions for circuit timing expectations can be used to determine how long information needs to be stored in memory, which determines the minimum memory required for the circuit. This information can be used to create a circuit configuration that maximizes functionality for a given set of resource constraints.

The second goal of this research was to make improvements to the design that increase downlink efficiency by taking advantage of the conjugate-symmetric property associated with the FFT of real signals. Using this property, the amount of FFT output information required to reproduce a real input signal can be reduced in half. Adjustments to the memory allocation and control algorithms were made to remove inefficiencies discovered though the circuit analysis and to implement savings using the FFT conjugate-symmetric property. The downlink algorithms were also adjusted to improve signal flow to an external communication system.

The resulting circuit was configured for a Virtex™-IIP target device. System Generator was used to create a Xilinx ISE project. Xilinx ISE Project Navigator software was used to synthesize the design, which produced a resource estimation confirming that

the design would fit within the resource constraints of the target device. In similar fashion, the design was configured for a Virtex™-I target device using the FFTv1.0 IP block. A multi-chip implementation was examined, where three Virtex™-I FPGAs connected in series were used. Simulation in MATLAB®/Simulink® and resource estimation provided by synthesis in Xilinx ISE Project Navigator confirmed that this is a feasible means of implementing the SDR design.

The final goal of this research was to add fault detection algorithms to make the design more suitable for the space environment. The FFT algorithm and temporary storage memory were identified as the most likely locations for faults within the design because of the large percentage of FPGA resources dedicated to each. The fault detection and correction methods of Triple Modular Redundancy (TMR) and Reduced Precision Redundancy (RPR) were examined. These methods were not used because employing them effectively would require more resources than were available on the FPGA.

Parseval's Theorem was used to relate the energy of the input signal to the energy of the output signal in a way that involves fewer computations than the FFT. After this method was implemented in the design, tests showed that the circuit was capable of detecting FFT errors. Parity checking was implemented to check for errors in temporary storage memory. Tests showed that the circuit would detect odd-numbered faults in each data word stored in memory. The downlink format was adjusted to communicate faults detected in either the FFT or memory to the end user.

The desired end state of this research was an improved version of the initial SDR design with increased compression and fault detection capability that can be loaded on an FPGA configured for space. This objective was met. The research demonstrated that a tactically relevant SDR design can be configured for use in the space environment.

To further develop this design toward use in actual spacecraft, the following additional research is required:

- Adjust the algorithm to compensate for a condition where the user defines two ROIs that overlap. This currently leads to inefficiency when information is transmitted twice – once for each ROI.
- Use pipelining features available in System Generator IP block interfaces to reduce the clock period. This will require adjustments to some of the circuit's timing algorithms.
- Use a larger range of input signals and user-defined configurations to test the design under more stressful conditions than those imposed in tests used for development. This should include using a broader range of fault injection experiments to test the sensitivity of fault detection algorithms.
- The user interface should be improved to prevent poor user-defined configuration choices and facilitate the set-up process.
- Explore other FPGA circuits to compute the FFT. This will provide more options and flexibility with ongoing circuit development. An FFT design using more elementary System Generator IP blocks could be used to implement internal fault detection and correction algorithms.

The ever-changing nature of threats to United States national interests generates an increasing demand for flexibility in space architecture. This design is an important part of on-going research to meet the demand through space-based FPGA designs.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This research would not have been possible without the insight, encouragement, and ingenuity of the NPS Configurable Fault Tolerant Processing team. Professor Frank Kragh was inspirational and always provided keen insight into the mathematics of signal analysis. Special thanks to Professor Herschel Loomis and Professor Alan Ross for their encouragement and expertise. No progress would have been possible without the outstanding technical support provided by Donna Miller on behalf of the Communications Research Laboratory.

Additional thanks to LT Durke Wright, USN, who got the ball rolling and provided excellent support from his follow-on assignment. Thanks to Capt Maggie Sullivan, USAF, for tenaciously demonstrating the role Space Systems students can have in the Electrical Engineering Department. Her example, advice, and encouragement were crucial to the successful completion of academic studies that contributed to this thesis development. Thanks to all fellow students in the Space Systems Engineering curriculum for unceasing encouragement and assistance in the baffling realm of Aeronautical Engineering.

Special thanks to all family members and friends for their love and support throughout the development of this thesis work. Thanks to all NPS faculty members and staff who provided support and encouragement. Thanks to the Trident Room staff for providing late evening sustenance on demand and sage-like bartender advice. Thanks to the members of the Saint Thomas Aquinas Chapel community for all of your prayers.

This thesis work would not have been possible without the dedication and sacrifices of all those who have answered the call to defend freedom. Special thanks to all those deployed to Iraq and Afghanistan during the development of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The acquisition of satellite systems is a process that typically takes several years between the identification of a need to the delivery of a capability. Operationally Responsive Space (ORS) is a strategy that strives to deliver space-based assets to the war fighter in a timely manner. Two important areas of concern are improving the flexibility of satellite designs and streamlining the acquisition process. The use of space-based Field Programmable Gate Arrays (FPGAs) can leverage both of these key ideas. [1]

An FPGA can be reprogrammed remotely to run different algorithms. A satellite with an FPGA could be reconfigured to meet a different mission on orbit, improving the flexibility of the asset. This also helps to streamline the acquisition process since FPGA design significantly reduces the amount of time involved with integration and testing. Reprogramming a space-based FPGA for a new mission completely bypasses the time, cost, and risk involved with launching a new satellite. [2]

An FPGA design with potential for space applications was presented in [3]. This initial SDR design is a signal analysis algorithm that compresses the output of a Fast Fourier Transform (FFT) computation. The design was created and tested at a high level of abstraction, using the Xilinx System Generator interface with the MATLAB®/Simulink® environment. It analyzes the information produced by the FFT computation to determine if the energy in user-defined frequency Ranges of Interest (ROI) meets user-defined thresholds. ROI that meet their threshold are forwarded to a downlink algorithm. ROI that do not meet their threshold are discarded. [3]

A. OBJECTIVES

This thesis work is intended to demonstrate the feasibility of using a space-based FPGA system as an option for ORS. This is accomplished by implementing a tactically relevant SDR algorithm in a configuration suitable for the space environment. The desired end state is an improved version of the design with increased compression and fault detection capability that can be loaded on an FPGA configured for space.

This thesis work produces an analysis of the hardware requirements associated with the initial SDR design. The information gained through this analysis is used to make changes to the design to ensure that it makes the best use of scarce FPGA memory resources. Improvements are also made to increase the efficiency of the data compression.

This research explores methods for improving the algorithm's fault detection capability. Fault detection is essential because it enables ground-based systems to assess the validity of the data produced. Fault detection is also necessary to determine when fault correction procedures are required, such as the reloading the FPGA configuration.

B. DESIGN APPROACH

This thesis work begins with an examination of the initial SDR design described in [3]. Simulations to examine the performance of the design and internal timing are conducted in the MATLAB®/Simulink® environment. The System Generator interface is used to generate the design in Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) format. Synthesis of the design from VHDL format is conducted using the Xilinx Integrated Synthesis Environment (ISE) Project Navigator. Post-synthesis resource analysis is conducted in the Xilinx ISE Project Navigator environment.

Modifications to the design discussed in Section B are made incrementally. After each change, the design is simulated using a baseline configuration and test set described in [3]. The effectiveness of each modification is evaluated based on the conformance of simulation results to expectations derived from the baseline test set. Any changes to the tests are noted where appropriate.

C. RELATED WORKS

This thesis work is primarily based on the initial SDR design work described in [3]. Although not specifically related to this work, several organizations are developing

space-based FPGA designs. One such design is the Programmable Satellite Transceiver presented in [4]. Wright describes other areas of research in signals analysis and the use of System Generator for FPGA design [3].

The work done to provide a fault detection capability in this design is closely related to the work described in [5] and [6]. Snodgrass presents a new method of error detection and correction called Reduced Precision Redundancy (RPR) [5]. This concept is discussed further in [7]. Sullivan expands on the work in [5] by examining the application of RPR to Digital Signal Processing and spacecraft attitude control [6]. This research, also described in [8], explores the feasibility of RPR for elementary algorithms then expands the scope to multi-level algorithms. This includes a study of using RPR to correct errors in the computation of the Fast Fourier Transform algorithm.

D. THESIS ORGANIZATION

This introductory chapter provides some background information and explains the objectives of the thesis work. The design approach is described and the thesis work is placed in the context of other related research.

Chapter II, Design Tools, discusses the resources that were used for the design. The chapter presents basic information about the FFT algorithm. The hardware and software tools selected for the design are listed. The chapter concludes with an analysis of the LogiCore FFTv1.0 and Xilinx FFTv4 Intellectual Property (IP) that were used to implement the FFT algorithm in hardware on Xilinx FPGAs.

Chapter III, Initial SDR Design, examines a SDR design that was introduced in [3]. The chapter presents graphical representations of certain design elements to improve understanding of their functionality. The chapter introduces an analysis of the design's timing and resource requirements. The chapter concludes by presenting generalized expressions for the circuit's latency and memory requirements based on the circuit's configuration and user-defined constraints.

Chapter IV, Initial Modifications to the Original Design, presents changes to the initial SDR design that make it more feasible for use on a resource-constrained FPGA.

The conjugate symmetry property of Fourier Transforms is used to increase downlink and memory efficiency. The header generation algorithm and Format Output subsystem are modified to improve efficiency and the ability to send signals to an external communications system. Modifications are made to accommodate the FFTv1.0 IP, enabling use on a Virtex™-I FPGA. The effectiveness of these changes is demonstrated by synthesizing the design for both a Virtex™-I FPGA and a Virtex™-IIP FPGA, producing an estimate of resource utilization for each.

Chapter V, Fault Detection, introduces methods that would enable the design to recognize when errors occur in either FFT computing or in temporary memory storage. After reviewing the feasibility of various redundant algorithms, a method using Parseval's theorem to compare the FFT input with the FFT output is explained. The Parseval-related method is demonstrated as a resource-efficient means of detecting errors in the FFT computation. Parity checking is demonstrated as a means to detect errors in temporary memory storage.

Chapter VI, Conclusions discusses the significance of the information presented in this thesis. The chapter also presents recommendations for future work.

II. DESIGN TOOLS

Implementing and analyzing the SDR requires the use and understanding of certain mathematical concepts and design tools. This chapter presents some basic information regarding Fourier analysis and FPGA design using the Xilinx System Generator tool. It also discusses some design considerations regarding the specific hardware and IP circuitry used. This chapter shows how the theory presented is implemented in hardware.

A. FOURIER ANALYSIS

The use Fourier analysis to examine the characteristics of signals is discussed in [3]. This section reviews the process and adds more information regarding the efficient implementation of Fourier analysis using FFTs, as discussed in [9**]**.

The Fourier Transform (FT) is used to analyze signals in the frequency domain. As discussed in [9], the frequency domain representation of the time domain signal $x(t)$ is

$$X(f) = \mathrm{FT}\{x(t)\} = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}dt \,. \qquad \text{(III.1)}$$

The frequency domain representation of the signal can be converted back to the time domain using the Inverse Fourier Transform (IFT),

$$x(t) = \mathrm{IFT}\{X(f)\} = \int_{-\infty}^{+\infty} X(f)e^{j2\pi ft}df \,. \qquad \text{(III.2)}$$

As shown in Equations II.1 and II.2, the Fourier transform and inverse Fourier transform are dependent on an infinite set of continuous samples**.** The Discrete Time Fourier Transform (DTFT), which provides a means for calculating the frequency representation of a discrete time sequence of infinite length is also discussed in [9]. The variable *t* is replaced with the variable *n*, indicating that the points are sampled discretely. The variables are related by the expression $t = nT$ where *T* is the sampling period and *n*

is required to be an integer. The sampled signal is $x[n] = x(nT)$. The variable *n* is used in the expression

$$X(f') = \mathrm{DTFT}\left\{x[n]\right\} = \sum_{-\infty}^{+\infty} x[n] e^{-j2\pi f' n} \quad . \tag{III.3}$$

In this expression, *f*' is the frequency in units of cycles per time index. For frequencies $f \in [-0.5 f_s, 0.5 f_s]$, $X(f) = X(f' f_s)$ if there is no aliasing and the sampling frequency is $f_s = 1/T$. Similarly, the Inverse Discrete Time Fourier Transform (IDTFT) provides the discrete time sequence associated with the frequency domain representation, as shown in the expression

$$x[n] = \mathrm{IDTFT}\left\{X(f')\right\} = \int_{-1/2}^{+1/2} X(f') e^{j2\pi f' n} df \quad . \tag{III.4}$$

As with the Fourier transform, the DTFT requires the analysis of a signal over all time. The IDTFT requires the analysis of an infinite number of frequencies. As discussed in [9], the frequency domain representation of a time domain signal can be estimated from a finite set of samples of length *N* using the discrete Fourier transform as shown in the expression

$$X[k] = \mathrm{DFT}\left\{x[n]\right\} = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \text{for } k = 0,...N-1 . \tag{III.5}$$

In this expression, $k = f'N$. The variable *k* is related to the continuous frequency *f* as shown in the expression

$$f = \frac{k}{N} f_s . \tag{III.6}$$

The inverse discrete Fourier transform reverses the process, producing the time domain signal of length *N* from the frequency domain signal defined on the set of *N* discrete frequencies, as shown in the expression

$$x[n] = \mathrm{IDFT}\left\{X[k]\right\} = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N}, \text{for } n = 0,..., N-1 . \tag{III.7}$$

To simplify these expressions the phase factor w_N is used. This element is also sometimes referred to as a "twiddle factor," defined by the expression

$$w_N = e^{-j2\pi/N}. \tag{III.8}$$

The straightforward DFT calculation for each element of the discrete frequency vector $X[k]$ requires N complex multiplication and addition operations. The computation of the entire set requires N^2 operations. The FFT is a means of calculating the same result by breaking up the expression in a way that uses fewer operations.

The radix-2 FFT implementation divides the DFT expression into its even and odd components, as shown in the equation

$$X_N[k] = \sum_{m=0}^{N/2-1} x[2m] w_N^{2mk} + w_N^k \sum_{m=0}^{N/2-1} x[2m+1] w_N^{2mk}. \tag{III.9}$$

The DFT calculation can be conducted in $N \log_2 N$ operations. [9]

B. COMPUTING TOOLS

Several computational tools were used to design, test, and implement the SDR in hardware. Wright discusses the FPGA design process in detail [3]. This section discusses the configuration of these tools, highlighting specific details critical to reproducing results described in the remainder of the thesis.

1. System Generator

As discussed in [10], System Generator is a hardware design tool produced by Xilinx. It produces Intellectual Property (IP) based pre-designed circuitry in a format that can be inserted in a signal flow path in the MATLAB®/Simulink® environment. This IP circuitry is made available through the Xilinx ISE software package. Circuit designers who wish to use System Generator are encouraged to use the training package available with the software. The package is labeled as a series of labs (1-7), located in the following path:

\Xilinx\10.1\DSP_Tools\sysgen\examples\getting_started_training

The MATLAB®/Simulink® environment offers several advantages when designing a circuit to be used for signal analysis. The computation tools in MATLAB® provide a means of easily generating input signals and interpreting output. The circuit diagram is easy to construct and visualize. Using the System Generator interface, the design can be compiled to any level from the Hardware Description Language (HDL) netlist down to the bitstream file required to program the target FPGA.

One disadvantage of designing in System Generator is that the extra layer of abstraction from the IP blocks disables some of the configuration options that would otherwise be available. System Generator is not a standard tool used in circuit design at this time, so many IP blocks are not available to be implemented in this environment. The way to work around these shortcomings is to use System Generator for high-level design and testing, and then compile the circuit to the HDL netlist level. From there, other design tools can be used to configure the circuit and design interfaces to other IP blocks.

2. Xilinx ISE

The Xilinx ISE Design Suite is another tool that was used in this design to configure the HDL netlist produced by System Generator. The Xilinx ISE interfaced with the ModelSim® simulator to confirm the results of tests that were conducted in the MATLAB®/Simulink® environment.

3. Interface

In order for the design environment to function, all software listed in this section must work together. The versions of each piece of software used for this design are shown in Table 1.

Software	Version	Purpose
MATLAB®/Simulink®	7.4.0 (R2007a)	Configure the design. Generate input. Interpret output.
Xilinx ISE Xilinx System Generator	10.1	Define the design structure and configure components.
ModelSim®	SE 6.3g	Simulate the design

Table 1. Development Software.

After installing all components, the Xilinx libraries must be compiled for ModelSim®. This can be accomplished by entering the *compxlib* command in the prompt available in Xilinx ISE Project Navigator under the "TCL Shell" tab. The command must be entered with the parameters shown in Figure 1. Although divided between two lines for easier reading, the command is entered without a carriage return until the end.

```
% compxlib -s -mti_se -arch all -lib all -l all -dir C:\Xilinx92i
-log compxlib.log -w -p C:\Modeltech_6.3g\win32 -smartmodel_setup
```

Figure 1. Compiling Instructions [After 11].

C. TARGET DEVICES

This design is intended for the Xilinx Virtex™ FPGA family. This section presents the capabilities of three devices within that family, specifically focusing on the available block memory. Memory is a critical constraint for the SDR design, which can be tailored for any of these devices. The design's capabilities and limitations change, depending on the target device. This will be discussed in greater detail later in this document.

Table 2 illustrates the memory capacities for three different Xilinx FPGA devices. The Virtex™-I device was selected as a baseline FPGA used in several legacy space systems. The Virtex™-II Pro device was selected for comparison because it is the

earliest model that is capable of supporting the FFTv4.1 IP circuit. The Virtex™-4 device was selected because it was a readily available target device used for the initial SDR design.

FPGA Series	Device	Block RAM (BRAM)	Blocks	Description
Virtex™-I	xcv1000	16 kB	32	512-Byte Blocks
Virtex™-II Pro	xc2vp20	1584 kB	88	18 kB Blocks
Virtex™-4	xc4vlx25	1296 kB	72	18 kB Blocks

Table 2. FPGA Memory [From 12–14].

D. FOURIER TRANSFORM COMPUTING

The System Generator toolbox provides several IP blocks that compute the FFT. Two of these blocks were selected as feasible candidates for computing the FFT for this design: Fast Fourier Transform v1.0 (FFTv1.0) and Fast Fourier Transform v4.1 (FFTv4.1). This section discusses the advantages and disadvantages of each IP block, and demonstrates the functionality of each as applied to the design.

1. Fast Fourier Transform v4.1

Within the Virtex™ FPGA family, FFTv4.1 is designed to work with the Virtex™-5, Virtex™-4, and Virtex™-II Pro. It provides the capability for pipelined, streaming I/O, which permits the continuous processing of data. For this reason, the pipelined configuration is preferred if using the FFTv4.1 with this SDR design. The FFTv4.1 can be configured to compute an FFT of any length N for $8 \le N \le 65536$ where N is a power of two [15].

a. Configuration Details

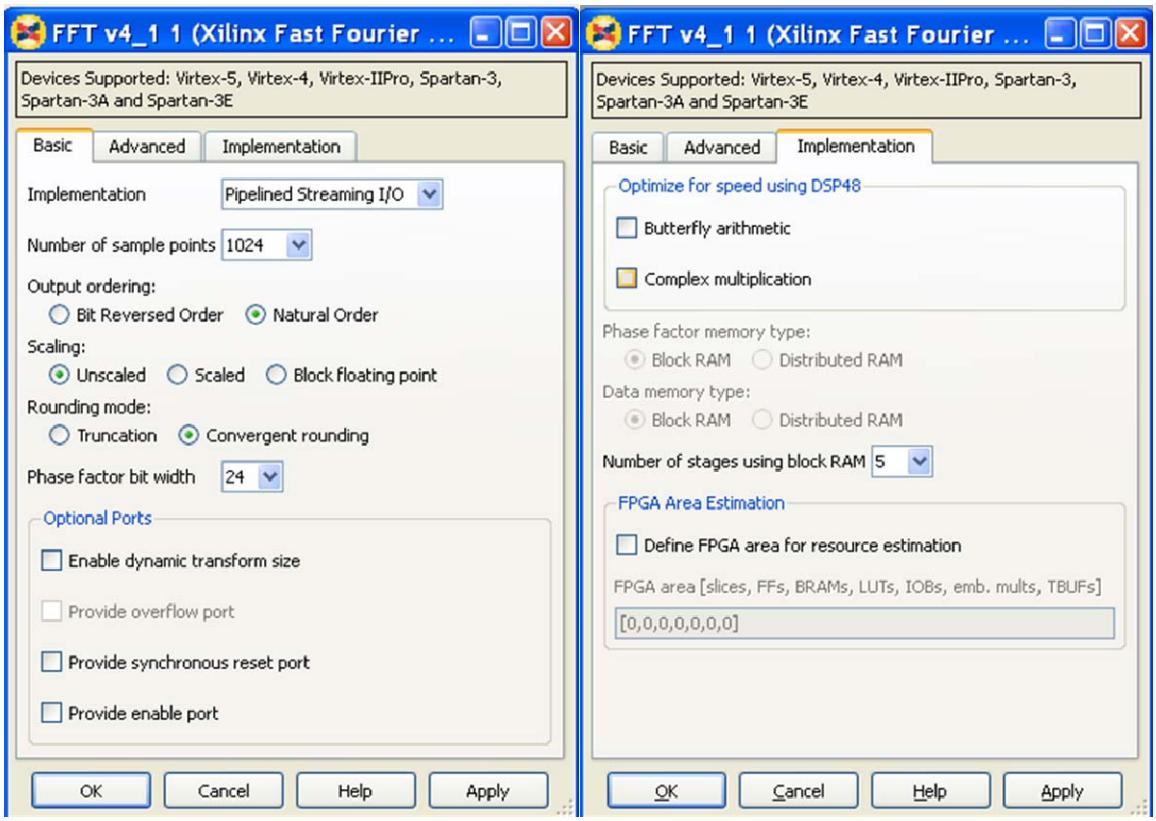

Figure 2. FFTv4.1 Implementation [After 16].

Figure 2 shows the configuration options for FFTv4.1 and displays those options that were selected. The circuit produces natural order output, which costs additional circuit resources and delay over the bit reversed output. Wright describes the input and output signals of this circuit in detail [3].

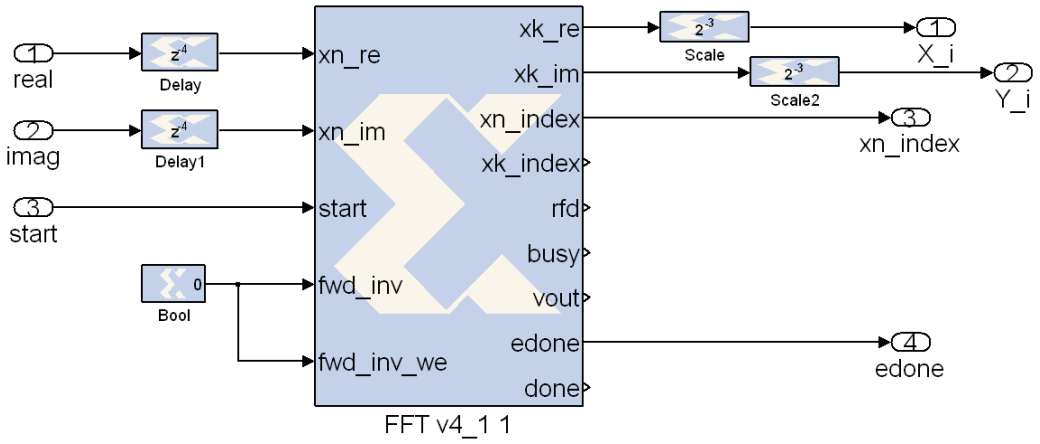

Figure 3. FFTv4.1 IP Block, $N = 8$ [From 3].

The crucial details of this circuit are its input format, output format, and timing. The input signal must already be separated into its real and imaginary components. For the purposes of this design, all signals are assumed to be real, so the imaginary component is hard-wired to binary 0. The input data must be in fixed point format, with the binary point at $D-1$ for a data word of width D. The output of the circuit has a data word width of $D+\log_2 N+1$ bits, which prevents overflow while maintaining the fixed binary point in the same location. As discussed in [15], the input data must be scaled such that $\left|x[n]\right|<1$. This configuration uses scaling modules outside the FFTv4.1 circuit to reduce the output by a factor of $1/N$, as shown in Figure 3.

b. Circuit Timing

As discussed in [3] and [15], the input signal '*start*' indicates that the FFTv4.1 circuit should begin accepting input. If *start* is asserted at time $t=0$, the FFT signals its readiness for input by asserting the '*ready*' flag and *xn_index* begins counting from 0 to $N-1$, incrementing on each clock. When $t=4$, the FFTv4 circuit accepts the first input, $x[0]$. At $t=1025$, *xn_index* restarts at zero and at $t=1028$ the circuit accepts the first sample of the next set of input data. There is no delay between the input of sequential data sets.

The signal *e_done* is asserted at $t = 2160$, indicating that the circuit will produce the first $X[k]$ output on the next clock cycle. At $t = 2161$ the signal *done* is asserted, *xk_re* and *xk_im* indicate the real and imaginary portions of $X[0]$ respectively, and *xk_index* begins counting from 0 to $N-1$, incrementing on each clock.

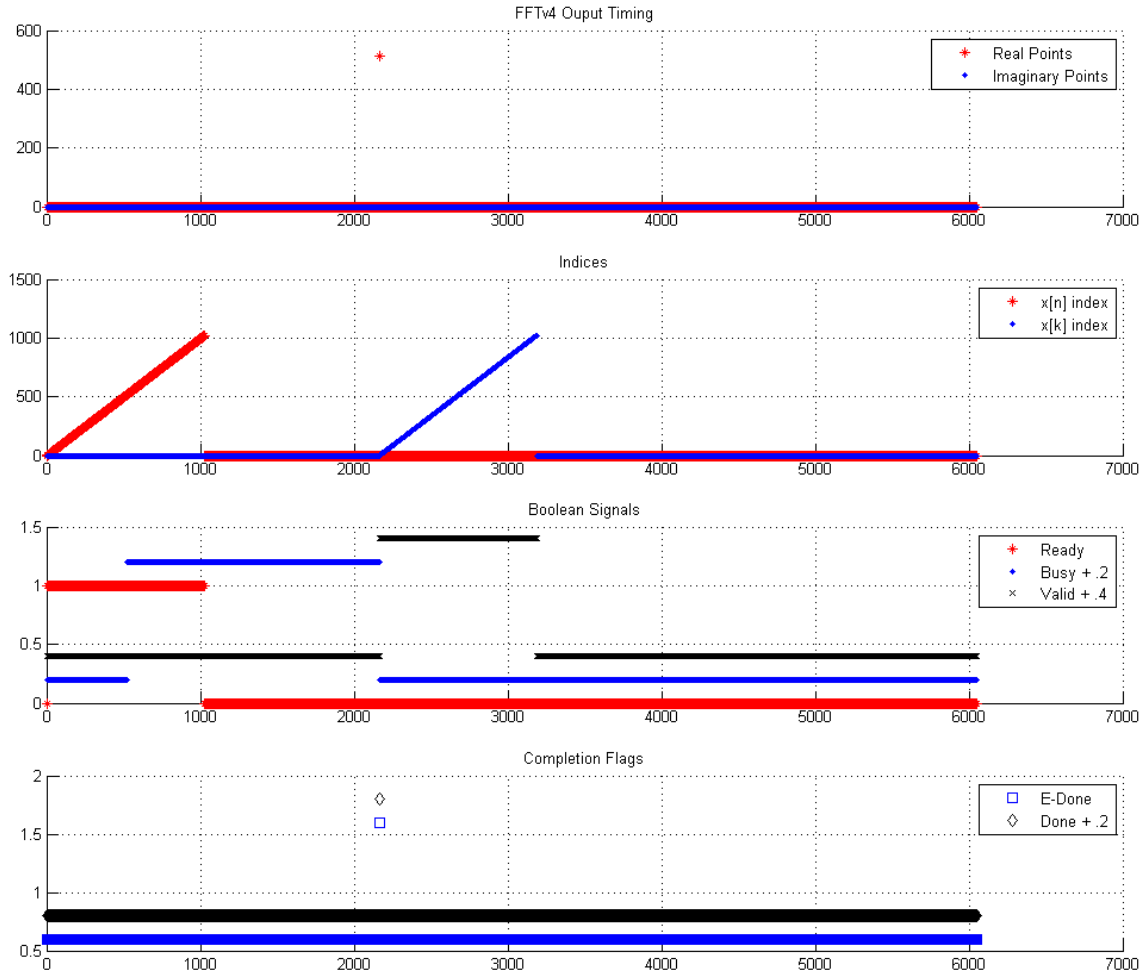

Figure 4. FFTv4.1 response to DC input.

The number of clock cycles between the last input point and the first output can be expressed as the latency $L = N + 8$ clock cycles. There is no delay between the outputs of sequential data sets. The timing of the FFTv4.1 circuit in response to a real, Direct Current (DC) signal ($x[n] = 0.5 \forall n$) is illustrated in Figure 4. The output displayed has not yet been rescaled, so the desired result is shown in Equation

(III.10). In this example, the *start* signal was only set for one clock cycle. After computing the first set of N points, the circuit stops accepting input.

$$X[0] = \sum_{n=0}^{N-1} x[n]e^{0} = Nx[n] = 1024 \cdot 0.5 = 512$$
$$X[k] = 0 \quad \text{for } k \neq 0 \tag{III.10}$$

The test was re-run with the *start* signal set high for $4 < t < 2N + 4$, with the results shown in Figure 5. This illustrates the circuit's ability to accept streaming input and produce streaming output.

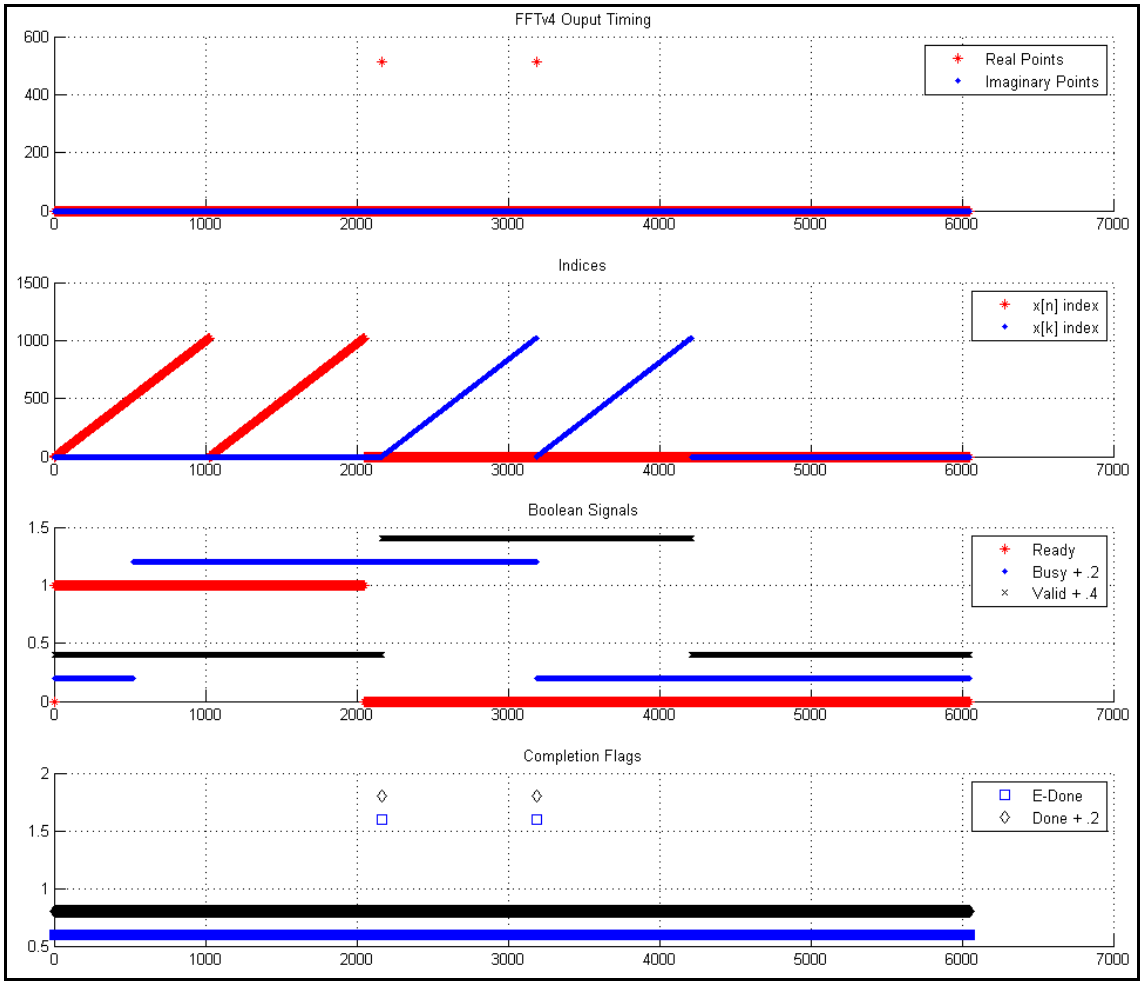

Figure 5. FFTv4.1 response to streaming DC input.

c. Resource Utilization

Table 3 illustrates the resources that the FFTv4.1 uses when the target device is the Virtex™-4 FPGA and the configuration options are selected with $N = 1024$, as shown in Figure 2. This confirms the estimation provided in [15]. The specific FPGA selected for this synthesis was an xc4vlx25-10sf363 model. The resource estimation was produced by the Xilinx ISE Project Navigator following synthesis. As discussed in [17], two slices are used to make a single Configurable Logic Block (CLB) on an FPGA. Four-input Look-Up Tables (LUTs) are function generators capable of implementing any Boolean function of four inputs. "Bonded IOB" indicates the number of Input/Output Blocks that were used. As discussed in [18], "FIFO16/RAMB16" indicates the number of 18kB Random Access Memory (RAM) blocks that were used. "DSP48" indicates an arithmetic primitive used for digital signal processing that consists of an 18-bit by 18-bit multiplier followed by a three-input adder. As discussed in [19], "GCLK" indicates the number of global clock buffers that were used.

Resource	Used	Available	Percent Used
Slices	4950	10752	46%
Flip Flops	8654	21504	40%
4 input LUTs	6606	21504	30%
Bonded IOBs	132	240	55%
FIFO16/RAMB16	20	72	27%
GCLK	1	32	3%
DSP48	48	48	100%

Table 3. FFTv4.1 Resource Utilization on a Virtex™-4 [From 20].

Table 4 illustrates the resources that the FFTv4.1 uses when the target device is the Virtex™-IIP FPGA and the configuration options are selected with $N = 1024$, as shown in Figure 2. This information confirms the estimation provided in [15]. As discussed in [21], "MULT 18X18" indicates the number of 18-bit by 18-bit multiplier primitives available. As discussed in [15], the model must have at least as

many resources as those available in the xc2vp20 series. The specific FPGA selected for this synthesis was an xc2vpx20-5ff896 model. In this configuration, the phase factor bit width cannot be greater than 16.

Resource	Used	Available	Percent Used
Slices	3734	9792	38%
Flip Flops	6459	19584	32%
4 input LUTs	4592	19584	23%
Bonded IOBs	115	552	20%
BRAM	16	88	18%
MULT 18X18	22	88	25%
GCLK	1	16	6%

Table 4. FFTv4.1 Resource Utilization on a Virtex™-IIP [From 20].

2. Fast Fourier Transform v1.0

From the family of Virtex™ FPGA devices the FFTv1.0 is designed to work only on the Virtex™-I. Although it does not provide the capability for a pipelined architecture with streaming output, it does provide the capability to sample continuously. When using the triple memory configuration, the circuit compensates for a latency of *3N* for each computation by sampling at one quarter of the clock rate ($f_s = f_c / 4$). This is true for any value of *N*. [22]

As discussed in [22], the FFTv1.0 is limited to 16-bit arithmetic. As with FFTv4.1, the input data must be scaled such that $|x[n]| < 1$. Unlike the FFTv4.0, the word format of the input propagates to the output. To prevent overflow, the output is automatically scaled by $1/N$ [22]. The circuit operation differs slightly between the features offered by the Xilinx FFTv1.0 core and its realization in the System Generator environment. These differences are noted in the following description where appropriate.

a. Configuration Details

As shown through a review of [16] and [22], there are differences between the HDL implementation of the FFTv1.0 IP block and its representation in the System Generator environment. Although FFTv1.0 provides the same output signals available from FFTv4.1, some of them are not available in the System Generator representation of the circuit. The signals *e_done*, *xn_index*, *xk_index*, and *busy* are not listed as available outputs, as illustrated in Figure 6. Their absence is not a major obstacle to development, since the signals used for this SDR design can be replicated using other features. By inserting one-clock delays in all other signal paths, the *done* signal can function as the *e_done* signal. The indices can be replaced by counters when synchronized with the *valid* and *done* signals.

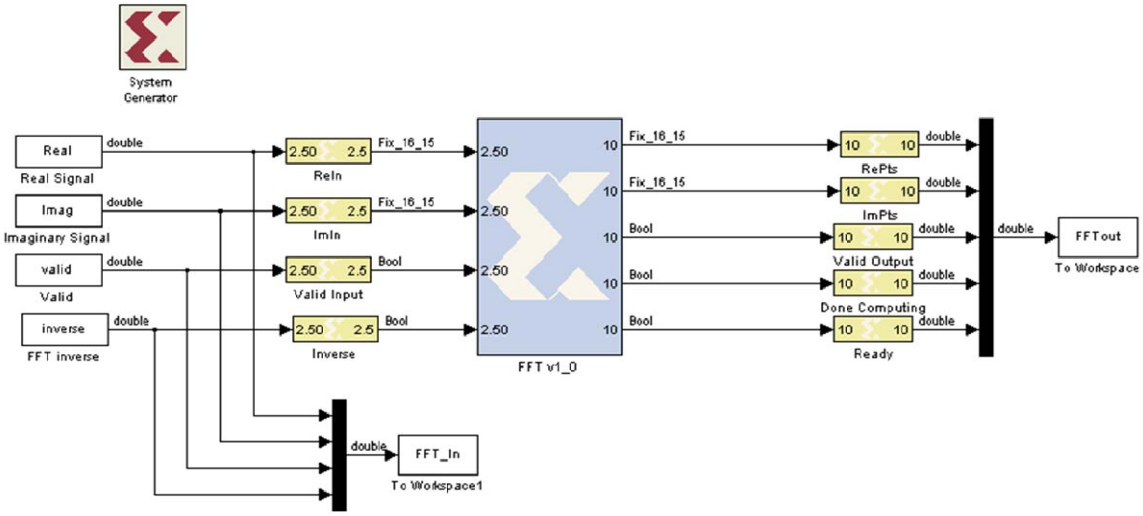

Figure 6. FFTv1.0 Test Circuit.

The FFTv1.0 circuit offers fewer configuration options in the System Generator Graphical User Interface (GUI). The System Generator user can only select a FFT length N such that $N \in \{16, 64, 256, 1024\}$, as shown in Figure 7. This suggests that the IP block uses radix-4 computations because each allowable size of N is a power of four. The "Memory Usage" option indicates how many N length buffers are used to manage the data flow within the design.

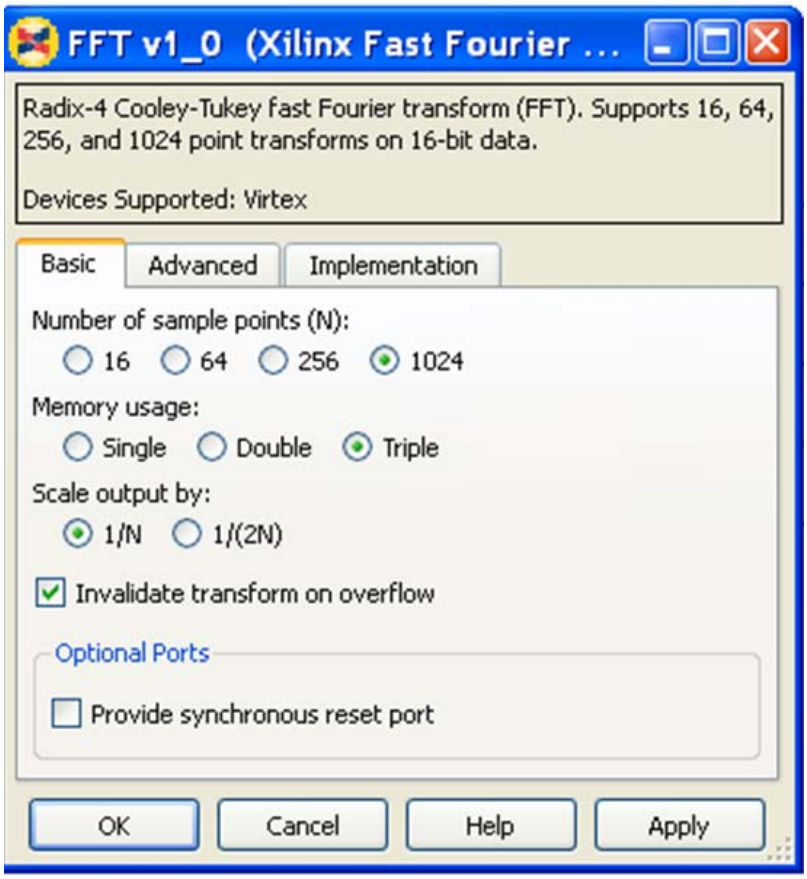

Figure 7. FFTv1.0 System Generator Configuration Options [After 16].

An abstract model of the "Triple Memory" configuration is shown in Figure 8. This architecture uses one buffer to store intermediate results during FFT computation. An output buffer allows the previous computation to be stored during output while the current one is being processed. An input buffer allows a third set of points to be sampled during the computation.

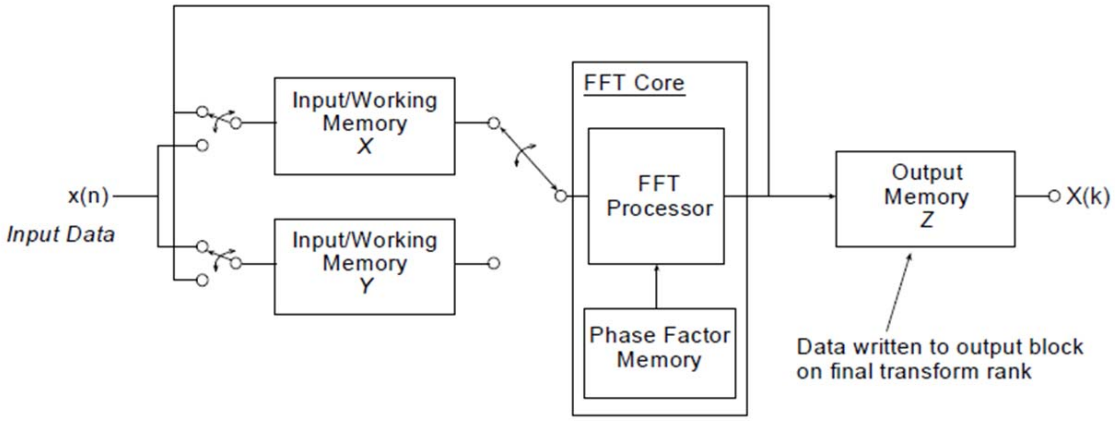

Figure 8. FFTv1.0, Triple memory configuration [From 22].

b. Circuit Timing

The FFTv1.0 circuit begins processing inputs when the *valid* signal is asserted. After the processing begins, a new set of 1024 (N) input points is sampled every 4096 clock cycles ($4N$). Although not used in this design, a synchronous reset port is available to stop the FFT processing. [22]

If the *valid* signal is asserted at $t = 1$, the *done* signal will be asserted at $t = 8246$. When done is asserted, the output signals *Xk_r* and *Xk_i* equal the real and imaginary components of $X[0]$. The output sequences from $X[0]$ to $X[1023]$ over 1024 clock cycles. During this time, the *vout* signal is asserted to indicate that the output is valid. On the next clock after $X[1023]$, the *vout* signal goes to zero for the next 3072 clock cycles, as the circuit computes the next FFT. The first point of the next data set is output at time 12342, 4096 clock cycles after the first point of the previous set. [22]

The FFTv1.0 circuit timing is demonstrated in a test run on a DC input signal. The *valid* input signal was asserted with $ReIn = 0.5$ at time $t = 4$. No input signals were changed for the duration of the test. The test results are shown in Figure 9. The figure illustrates that circuit produced the expected output,

$$\begin{aligned} \frac{X[0]}{N} &= \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{0} = x[n] = 0.5 \\ \frac{X[k]}{N} &= 0 \quad \text{for } k \neq 0 \end{aligned} \quad . \tag{III.11}$$

As anticipated, after the initial delay the output is valid for the first 1024 clock cycles out of every 4096. The *Ready For Data (RFD)* signal is asserted for all time. This indicates that the FFTv1.0 circuit is continuously sampling. [22]

After its first assertion, the *valid* signal indicates whether the sampled inputs are valid. De-asserting the *valid* signal indicates that the sampled input is invalid. If any portion of the sampled input is accompanied by an invalid signal, the FFTv1.0 circuit considers the entire input set to be invalid. In the corresponding output sequence, $vout = 0$. In the ISE/ModelSim® environment invalid output is displayed as, $X[k] = 0$ for all *k*. In the MATLAB®/Simulink®/System Generator environment, invalid output is displayed as *NaN*, indicating that the value cannot be computed as a number.

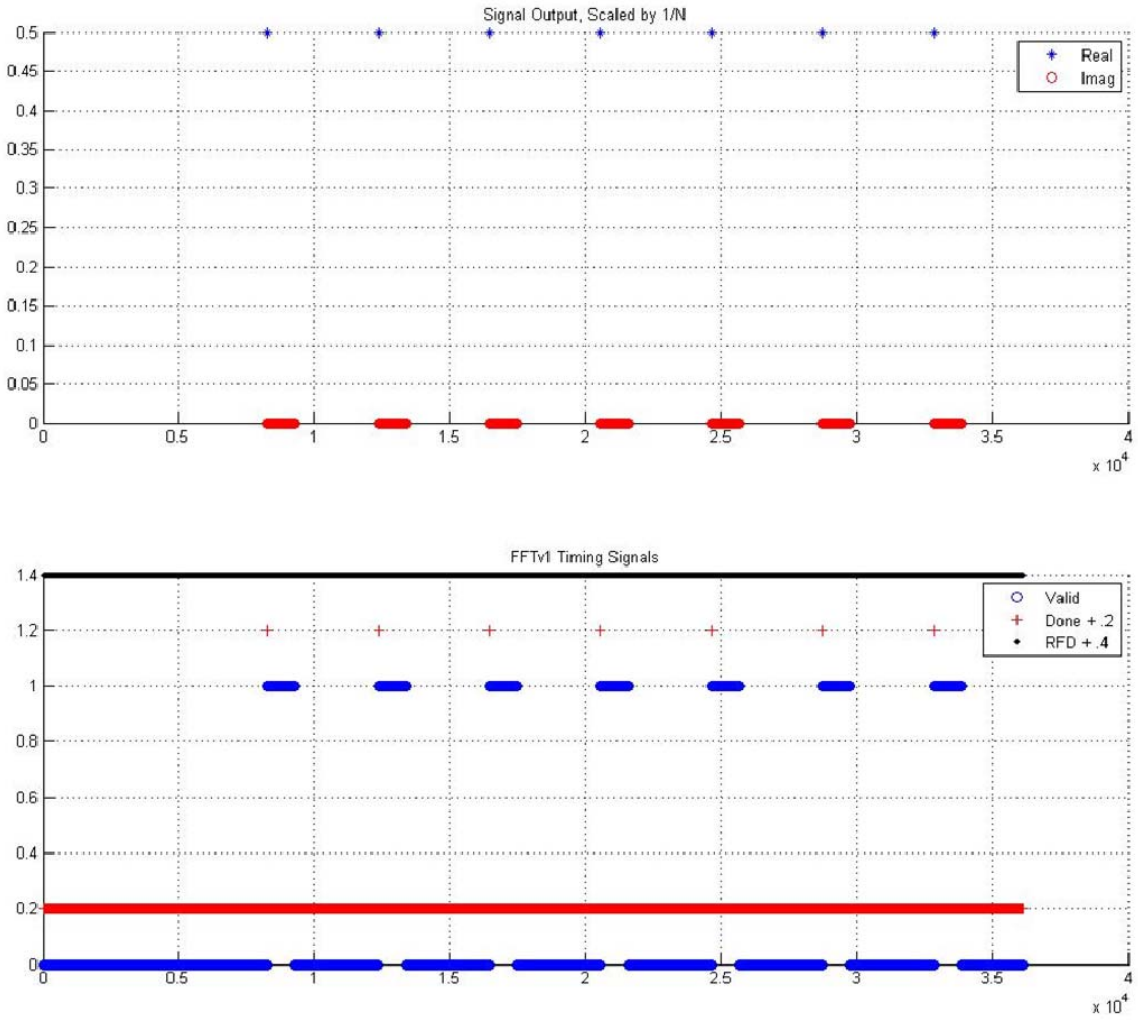

Figure 9. FFTv1.0 Response to DC input.

The performance of the *valid* input signal is demonstrated in Figure 10. The input is a real DC signal, where $x[n] = 0.5 \ \forall n$. The input *valid* signal is asserted at $t = 4$, initiating FFT sampling and processing. After one clock cycle, the input *valid* signal returns to zero. Since the FFT detects that the input is invalid, the corresponding output, indicated by the *done* signal asserted at $t = 8246$ is invalid, with *vout = 0*. In the next input sequence starting at $t = 5000$, *valid* is asserted for all samples except for one at $t = 6144$. Because one input point was invalid, the corresponding output, starting at $t = 12342$ is invalid. The input *valid* signal is asserted for the remainder of the test, and the first valid output is produced at $t = 16438$.

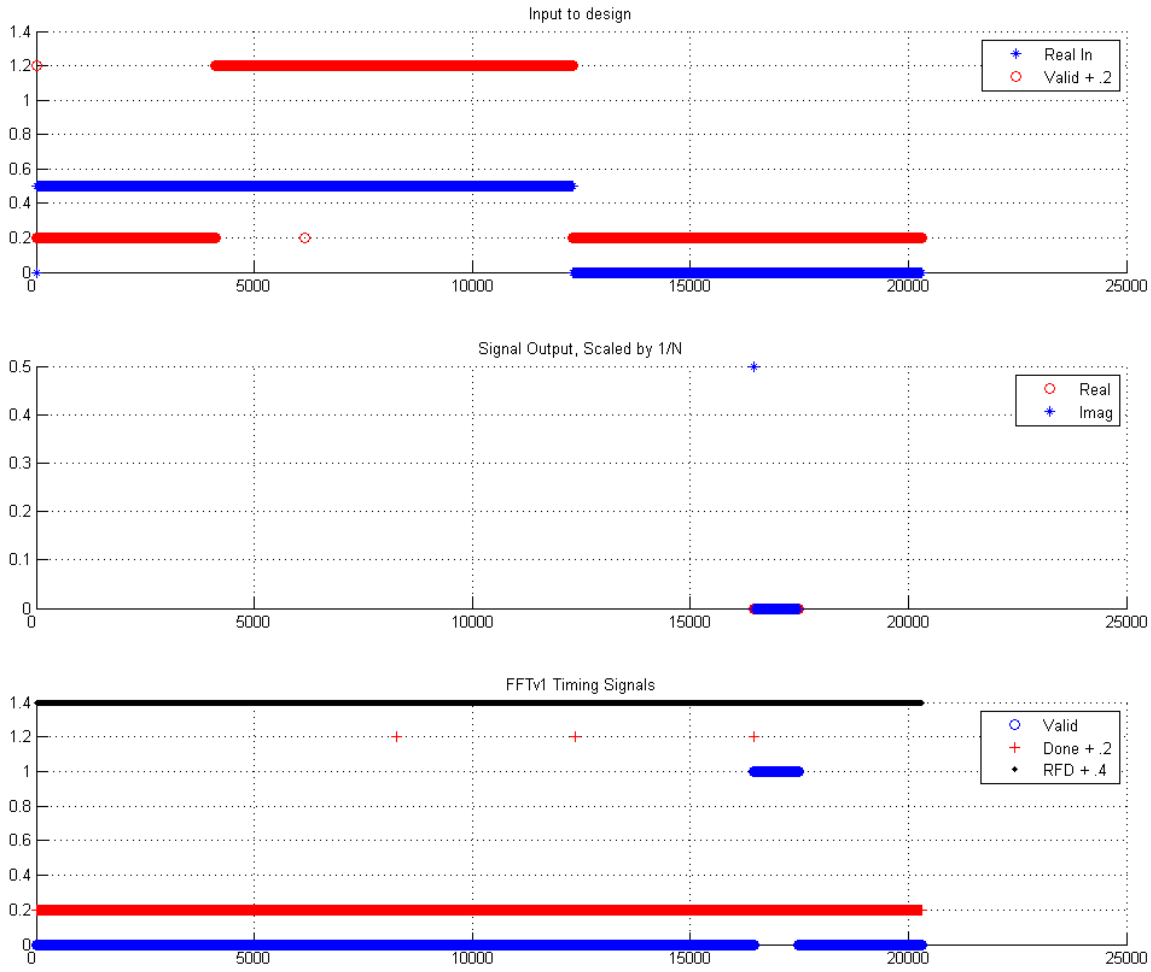

Figure 10. Performance of the *valid* signal.

c. Resource Utilization

Using the Triple Memory configuration discussed in this section, the FFTv1.0 circuit uses the resources shown in Table 5. The specific FPGA selected for this synthesis was an xcv1000-4bg560 model. The resource estimation was produced by the Xilinx ISE Project Navigator following synthesis. The FFTv1.0 IP was configured with $N = 1024$, as shown in Figure 7.

Resource	Used	Available	Percent Used
Slices	1289	12288	10%
Flip Flops	2577	24576	10%
4 input LUTs	2245	24576	9%
Bonded IOBs	70	404	17%
BRAM	24	32	75%
GCLK	1	4	25%

Table 5. FFTv1.0 Resource Utilization on a Virtex™-I [From 20].

d. Circuit Limitations

Designers using the FFTv1.0 circuit need to be conscious of the limitations on precision imposed by the combination of 16-bit computations, propagating the input signal format to the output, and the scaling of the output signal by $1/N$.

If a maximum-precision output is desired, its format must be a 16-bit number with the radix point at 15. Using this format, the smallest detectable output is

$$0.000000000000001_2 = \left(1/2^{15}\right)_{10} \approx 3.0513\times10^{-5}. \quad \text{(III.12)}$$

The input must be in the same format, so the smallest possible input signal is the same size if the full dynamic range of the input signal is used. Suppose an impulse signal is input to the FFTv1.0 circuit, where

$$\begin{aligned} x[n] &= 0.000001_2 = 2_{10}^{-6} = 0.0156 \text{ for } n = 0 \\ x[n] &= 0 \quad \text{for } n \neq 1\ldots N-1 \end{aligned}. \quad \text{(III.13)}$$

In this case, the corresponding output should be

$$\frac{X[k]}{N} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} = \frac{1}{N}\left(0.0156e^{o} + \sum_{n=1}^{N-1} 0 \times e^{-j2\pi kn/N}\right). \quad \text{(III.14)}$$

$$\frac{X[k]}{N} = \frac{0.0156}{1024} = \left(1.5234 \times 10^{-5}\right)_{10} = 0.0000000000000001_{2}$$

This is smaller than the minimum detectable output signal, so the output would appear to be $X[k] = 0 \forall k$.

In this configuration, if it is desired that all signals propagate through the circuit to produce valid output, then the input signal is limited to 6 bits with 5 bits representing the fraction. In this case, the smallest possible input value is $2^{-5} = 0.313$. The user may decide to use a larger range than this, but must be aware of the fact that not all signals will propagate through the circuit to produce a valid output.

E. CONCLUSION

This chapter discussed the computing resources required to implement the SDR design. The mathematical basis for the FFT was discussed. Software tools were introduced, as well as target FPGA devices. The chapter demonstrated the function of IP circuitry available to compute the FFT. The next chapter discusses the memory requirements and timing details of the initial SDR design.

III. INITIAL SDR DESIGN

This chapter reviews and augments the work documented in [3]. The referenced NPS thesis illustrates the performance of an SDR compression algorithm under various test scenarios. These demonstrations focused on the circuit's output as a function of the input. While this information is useful to the circuit's end user, it lacks a level of detail needed for future development of the circuit. This chapter traces the design's signal path, highlighting signals and configurations critical to the circuit's function. It also illustrates the intermediate responses of the system subcomponents as the signal propagates from the input to the output.

Although some versions of the original design were tested in hardware, a detailed resource analysis was not conducted. The most critical resource constraint for the SDR compression algorithm is the amount of memory available on the target FPGA. This chapter discusses the resource requirements of the original design. It also discusses the capabilities and limitations of the circuit, based on the resources available using various configurations and target devices. The chapter concludes with some general equations describing the circuit's overall performance and resource requirements. These equations are useful tools in configuring the circuit for optimal performance.

A. OVERALL FUNCTIONALITY

The initial SDR design calculates the FFT of a sampled, pre-demodulated, Intermediate Frequency (IF) signal. A conceptual model of the system is shown in Figure 11. The output FFT points are stored in temporary memory while the Bin Energy Calculation subsystem sorts the signal into time and frequency bins. This subsystem computes the amount of energy in each bin [3].

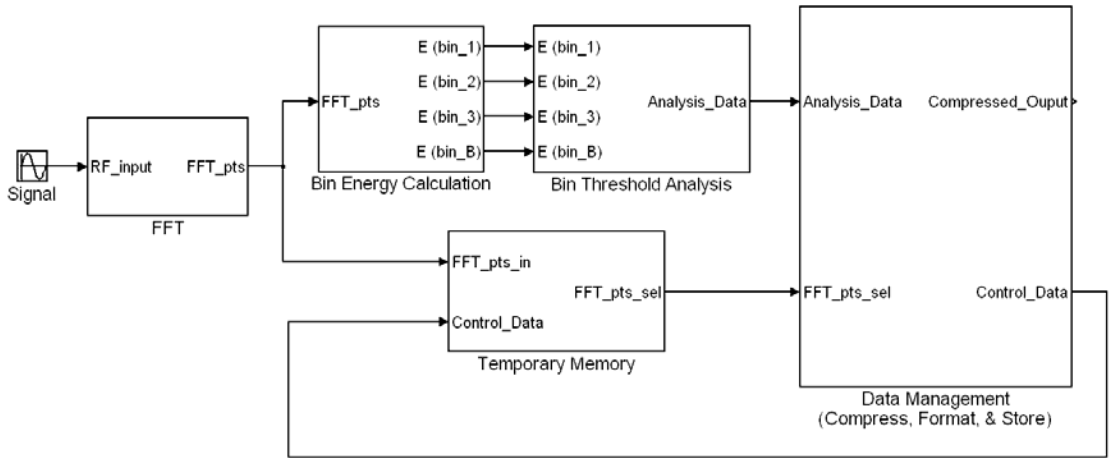

Figure 11. Conceptual SDR Model [From 3].

The Bin Threshold Analysis subsystem determines if the energy in each bin exceeds a user-defined threshold. The Data Management subsystem pulls FFT points from memory that correspond to bins exceeding the minimum energy threshold and adds header information to enable the reconstruction of the signal after downlink [3].

B. DESIGN SETUP

This chapter focuses on all circuitry after the calculation of the FFT. The majority of the flow path is linear, although a limited number of signals loop back, providing input to algorithms earlier in the flow path. No internal signals govern the function of the FFT circuit. The overall circuit design is displayed in Figure 12. This shows that the compression portion of the circuit reacts to stimuli from the FFT circuit. The FFT circuit only reacts to stimuli external to the circuit.

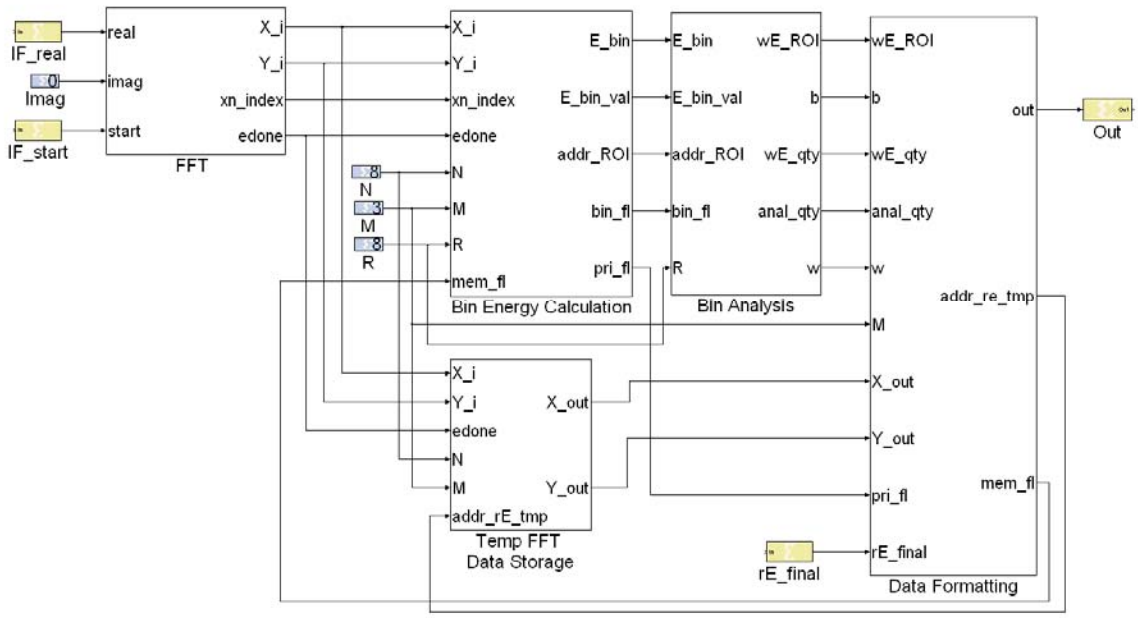

Figure 12. Overall Circuit Design [From 3].

The specific configuration discussed in this chapter uses the FFTv4.1 IP circuit. The FFT circuit computes the 1024-point FFT of a real signal. The data word format will be referred to using the notation *BitWidth_DecimalPoint*. The input signal had a width of 24 bits, with the decimal point to the left of bit 23. In other words, the data word format was 24_23. As discussed in Section 2.D.1, the output bit format is calculated to be

$$BitWidth_{output} = BitWidth_{input} + \log_2 1024 = 34\_23. \qquad \text{(IV.1)}$$

In the initial SDR design, the output of the FFT is then reformatted, scaling the signal by a factor of 2^{-10} and prepending a zero to the most significant bit. The signal that enters the Bin Energy Calculation block has a data word format of 35_33.

The *start* signal for the FFTv4.1 circuit is set at time *t = 1*. The first input point $x[0]$ is sampled at $t = 1$ and delayed four clock cycles, entering the FFT circuit at $t = 5$. After the initial latency discussed in Section 2.D.1 and a 48 clock delay, the first output point $X(0)$ reaches the Bin Energy Calculation block at time $t = 2209$.

C. BIN ENERGY CALCULATION

1. Time Windowing Subsystem

The function of the Time Windowing subsystem is to calculate the amount of energy in each FFT output point, then sum the amount of energy in each point over a time period determined by the user. The time period must be in integer multiples of the FFT period. The signal *M* is the number of FFT periods considered for each time window. [3]

a. Signal Flow

As discussed in [3], the energy in each point is calculated using the expression

$$\text{Energy}(k) = \text{Re}(k)^2 + \text{Im}(k)^2 . \qquad \text{(IV.2)}$$

This is implemented using two multipliers and one adder, as shown in Figure 13.

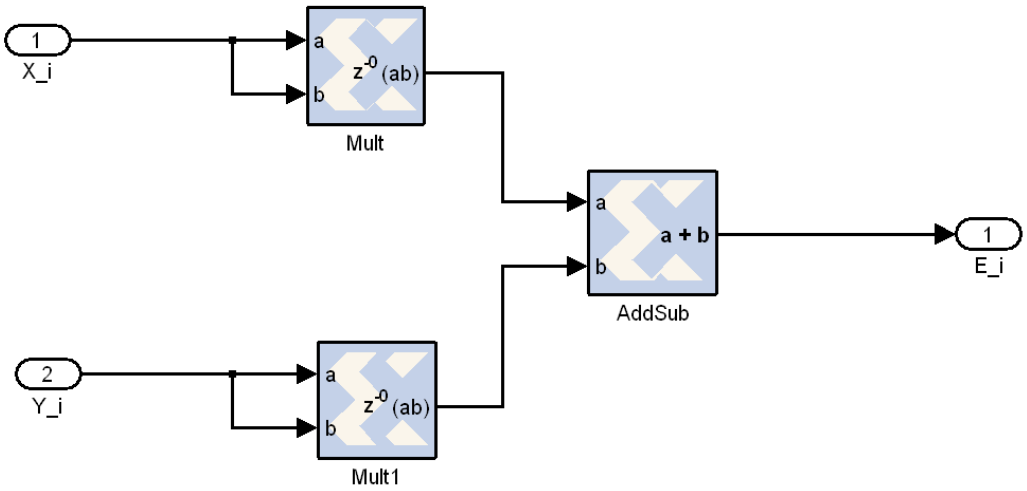

Figure 13. Circuit to Calculate Energy(*k*) [From 3].

System Generator provides a circuit designer with the option to set the output precision and latency of the addition and timing blocks. By double-clicking on the block, the designer can force the output data word into a desired format using a graphical interface shown in Figure 14. The latency can also be adjusted in terms of clock cycles.

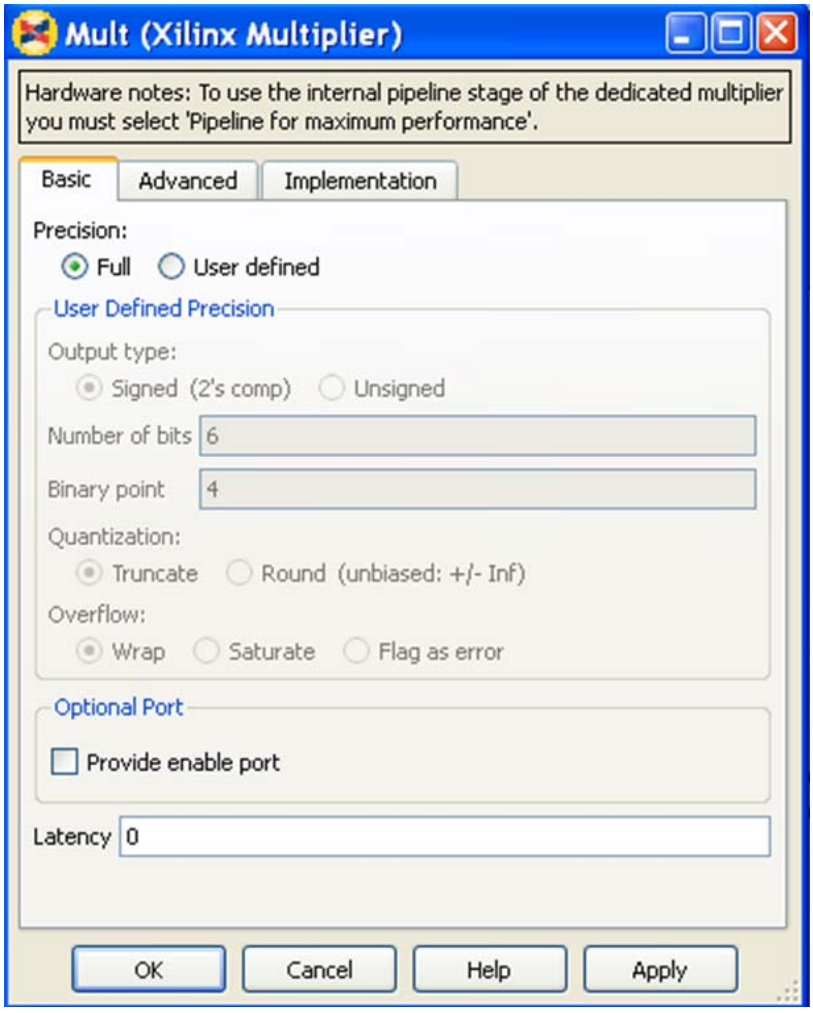

Figure 14. Multiplier IP Configuration [From 16].

Increasing the latency allows the compiler to implement the specified adder or multiplier circuit using the pipeline with the number of stages specified in the latency block. As discussed in [23], pipelining breaks up the circuit so that the longest delay path between registers is reduced. This allows the overall clock speed to be increased, which increases the sampling rate, increasing the highest analog frequency that the FFT can measure. No pipelining was used for any addition or multiplication blocks in the initial SDR design.

In the initial SDR design, the output precision for this energy calculation was set to "Full." In this configuration, bits are added to the output data word to preclude the possibility of overflow. The number of bits is doubled in each multiplier,

transforming the 35_33 input signal into an output signal with data word format 70_66. The full precision adder appends one bit to the signal, resulting in a data word format of 71_66.

The flow path for the time window energy calculation is shown in Figure 15. As discussed in [3], the energy signal is stored in a First-In, First-Out (FIFO) buffer. As successive sets of *N* output points are received from the energy calculation circuit, each new value is added to the corresponding point's energy accumulation from the previous time periods. The sum is written back into the FIFO buffer provided the number of FFTs processed is less than *M*, the number of FFT periods in a time window.

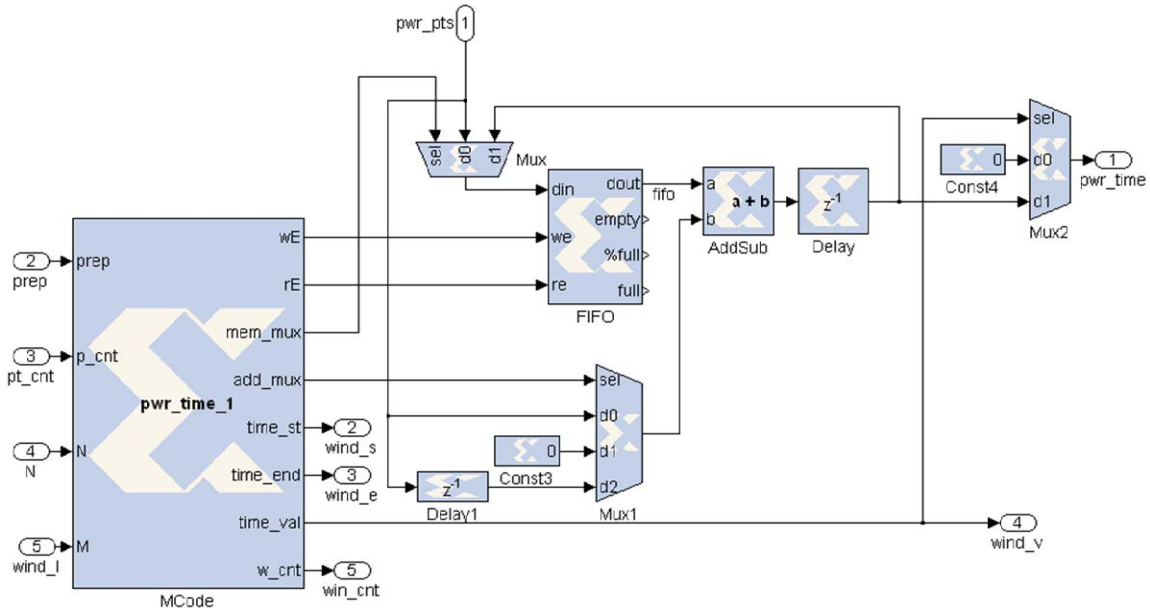

Figure 15. Time Window Energy Calculation [From 3].

The adder used for this circuit is configured with a latency of zero clock cycles, indicating that pipelining is not used. The output precision cannot be set to "Full" because the circuit's input is a function of its output. When System Generator attempts to compile the circuit in this configuration, it generates an error because it assumes size of the adder's output data word would grow without bound. In the initial SDR design, the adder's output precision was fixed at 54_42.

b. Timing Analysis

As discussed earlier, in the initial SDR design the latency of the adder and multiplier circuits is zero. The amount of time spent processing FFT points stored in the FIFO is dependent on the value of *M*. The circuit must wait until $N(M-1)$ points have been collected, with the summed energy vector calculated and stored in the FIFO.

When the Time Windowing subsystem receives the first point of the final FFT period within the window, the output of the adder can be forwarded to the next SDR subsystem. The circuit has an additional two-clock delay between the time an input is received and the time it is available for output, which accounts for the inherent delay associated with the FIFO buffer. For a case where $M = 3$, the first point of the third FFT period would enter the Time Windowing subsystem at time, $t = 2209 + 2N = 4257$. The first output point, indicating $|X_0(0)|^2 + |X_1(0)|^2 + |X_2(0)|^2$ leaves the Time Windowing subsystem at time, $t = 4257 + 2 = 4259$. The last output point, indicating the sum $|X_0(1023)|^2 + |X_1(1023)|^2 + |X_2(1023)|^2$ is forwarded to the Frequency Windowing subsystem at time, $t = 4259 + 1023 = 5282$.

The signal flow through this subsystem is managed by a Finite State Machine (FSM) implemented in M-Code by the *pwr_time* algorithm. The circuit's input and output signals are described in [3]. In addition, the circuit uses internal variables *fft_count*, *i_cnt*, and *delay_fl* to control state transitions and output timing. For this subsystem, the signal *p_cnt* is used to indicate the signal *xn_index*, from the FFT circuit. The signal *prep* is used to indicate the *edone* flag from the FFT circuit.

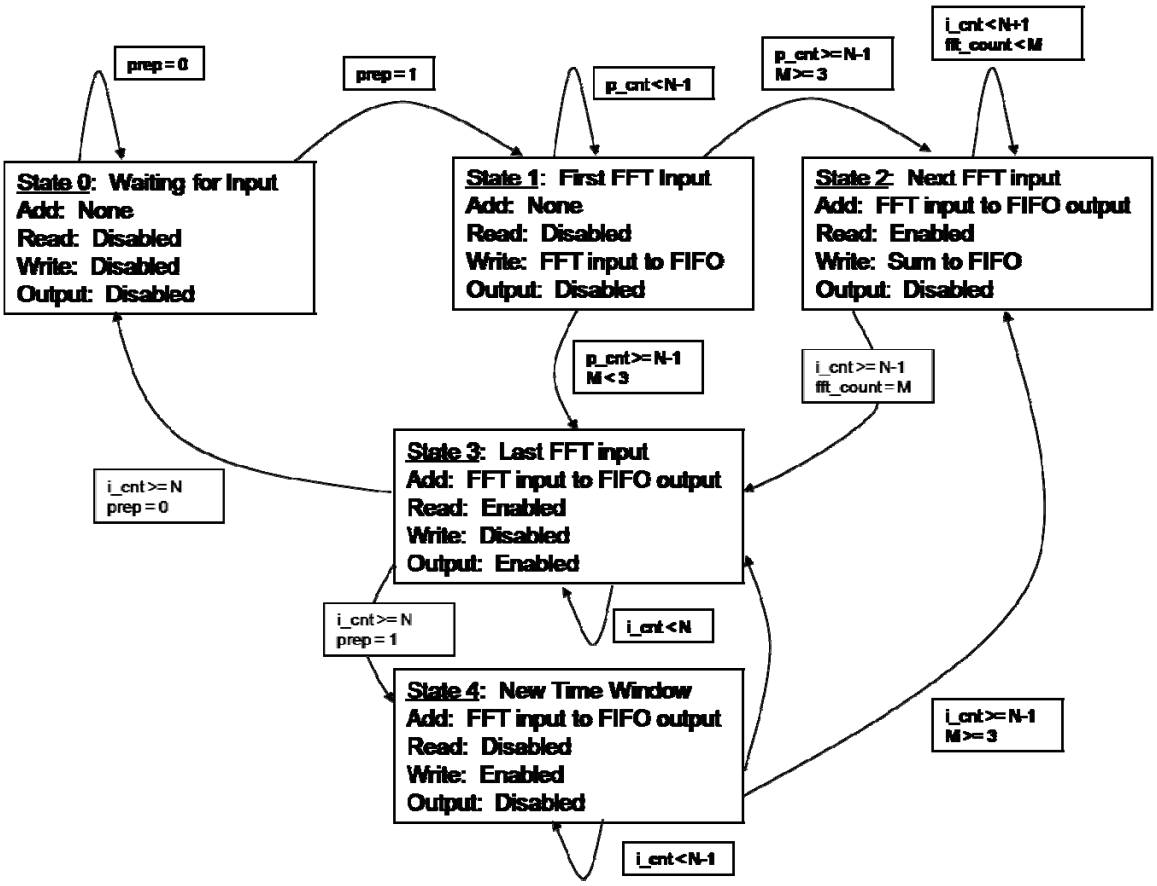

Figure 16. State Transition Diagram for *pwr_time* Algorithm.

A state transition diagram for the *pwr_time* algorithm is displayed in Figure 16. The state transitions and output for the *pwr_time* FSM are displayed in Moore format, with the output solely a function of the current state [23]. In truth, the FSM is in implemented in Mealy format, meaning that the output is a function of both the current state and the inputs [23]. When implemented in hardware, the internal variables *i_cnt*, and *delay_fl* would function as inputs to the FSM, changing the way each state produces output.

The affect of internal signals on the output for a case where $M = 3$ is illustrated in Figure 17. The multiplexer which controls the input to the FIFO buffer is governed by the *mem_mux* signal. The FIFO input is either from the FFT circuit, indicated by "FFT," or from the adder circuit, indicated by "Add." The multiplexer which controls the input to the adder is governed by the *add_mux* signal. The adder input

is either directly from the FFT circuit, indicated by "FFT," from the FFT circuit with a delay of one, indicated by "Del," or simply zero. The first two clock cycles of states two and three have different output, disabling the ability to write to the FIFO buffer until the adder output is ready.

										Sum (Time 1+Time2)							
Source	**Signal**		Time Window 1							Time Window 2							
Clock	Clock	2208	2209	2210	2211	...	3230	3231	3232	3233	3234	3235	...	4254	4255	4256	
FFT	p_cnt		0	1	2	...	1021	1022	1023	0	1	2	...	1021	1022	1023	
FFT	prep	1							1							1	
Internal	state	**0**	**1**	**1**	**1**	**...**	**1**	**1**	**1**	**2**	**2**	**2**	**...**	**2**	**2**	**2**	
Internal	i_cnt	0	0	0	0	...	0	0	0	0	1	2	...	1021	1022	1023	
Internal	delay_fl	0	0	0	0	...	0	0	0	1	2	2	...	2	2	2	
Internal	fft_count	0	1	1	1	...	1	1	1	2	2	2	...	2	2	2	
pwr_time	we	0	1	1	1	...	1	1	1	0	0	1	...	1	1	1	
pwr_time	re	0	0	0	0	...	0	0	0	1	1	1	...	1	1	1	
pwr_time	mem_mux	FFT	FFT	FFT	FFT	...	FFT	FFT	FFT	FFT	FFT	Add	...	Add	Add	Add	
pwr_time	add_mux	FFT	FFT	FFT	FFT	...	FFT	FFT	FFT	0	Del	Del	...	Del	Del	Del	
	FIFO in (k)	X	0	1	2	...	1021	1022	1023	X	X	0	...	1019	1020	1021	
	FIFO out (k)	0	0	0	0	...	0	0	0	0	0	1	...	1020	1021	1022	
	Adder In (k)		0	1	2	...	1021	1022	1023	0	0	1	...	1020	1021	1022	
	Adder Out		0	1	2	...	1021	1022	1023	0	0	1	...	1020	1021	1022	
	pwr_time (k)	0	0	0	0	...	0	0	0	0	0	0	...	0	0	0	
pwr_time	time_st	0	0	0	0	...	0	0	0	0	0	0	...	0	0	0	
pwr_time	time_end	0	0	0	0	...	0	0	0	0	0	0	...	0	0	0	
pwr_time	time_val	0	0	0	0	...	0	0	0	0	0	0	...	0	0	0	
pwr_time	win_cnt	1	1	1	1	...	1	1	1	1	1	1	...	1	1	1	
		Sum (Time 1+Time2+Time3)															
Source	**Signal**	Time Window 3															
Clock	Clock	4257	4258	4259	4260	...	5278	5279	...	5280	5281	5282	5283				
FFT	p_cnt	0	1	2	3	...	1021	1022	...	1023	0	1	2				
FFT	prep									1							
Internal	state	**2**	**2**	**3**	**3**	**...**	**3**	**3**	**...**	**3**	**3**	**3**	**4**				
Internal	i_cnt	1024	0	1	2	...	1020	1021	...	1022	1023	1024	1				
Internal	delay_fl	2	2	0	0	...	0	0	...	0	0	0	0				
Internal	fft_count	2	2	3	3	...	3	3	...	3	3	3	1				
pwr_time	we	1	1	0	0	...	0	0	...	0	1	1	1				
pwr_time	re	1	1	1	1	...	1	1	...	1	1	0	0				
pwr_time	mem_mux	Add	Add	FFT	FFT	...	FFT	FFT	...	FFT	FFT	FFT	FFT				
pwr_time	add_mux	Del	Del	Del	Del	...	Del	Del	...	Del	Del	Del	FFT				
	FIFO in (k)	1022	1023	X	X	...	X	X	...	X	0	1	2				
	FIFO out (k)	1023	0	1	2	...	1020	1021	...	1022	1023	0	0				
	Adder In (k)	1023	0	1	2	...	1020	1021	...	1022	1023	0	1				
	Adder Out	1023	0	1	2	...	1020	1021	...	1022	1023	0	1				
	pwr_time (k)	0	0	0	1	...	1019	1020	...	1021	1022	1023	0				
pwr_time	time_st	0	0	1	0	...	0	0	...	0	0	0	0				
pwr_time	time_end	0	0	0	0	...	0	0	...	0	0	1	0				
pwr_time	time_val	0	0	1	1	...	1	1	...	1	1	1	0				
pwr_time	win_cnt	1	1	1	1	...	1	1	...	1	1	1	2				

Figure 17. Timing of Output for *pwr_time* Algorithm, $N = 1024, M = 3$.

The FSM is designed for continuous, streaming input. This is the reason for State Four, which permits processing of the next time window while the last two

points from the previous window are output. After State One, all transitions are based on the internal variables *i_cnt* and *fft_count*. Regardless of the input received in successive FFT sets, the algorithm continues processing until it reaches the end of State Three. If the expected *prep* flag is not detected at this point, the algorithm returns to State Zero. Additionally the *time_st* and *time_end* flags are only set on the first and last clocks of State Three, respectively.

c. Memory Analysis

The biggest resource constraint for the Time Windowing subsystem is the amount of memory required for the FIFO buffer. The memory required is a function of the bit width of the data word and the depth of the memory. System Generator selects the bit width of memory based on the bit width of the input. The depth of memory, which indicates the maximum number of data words that can be stored, can be adjusted by the circuit designer using a pull-down menu, where

$$Depth = 2^n \quad \text{for } n = 4, 5, ..., 16 . \tag{IV.3}$$

After the values from the first N points are stored, the FIFO buffer reads a value on every clock cycle. Therefore, the FIFO buffer only requires enough depth to contain N values. The initial SDR design sets the FIFO depth to "4K." The corresponding memory requirement is

$$\begin{aligned} &\text{Memory} = \text{Bit Width} \times \text{Depth} \\ &\text{Memory} = 71 \text{ bits} \times 4\text{K} = 284 \text{ Kbit} = 35.5 \text{ KB} \end{aligned} . \tag{IV.4}$$

2. Frequency Windowing Subsystem

As discussed in [3], the Frequency Windowing subsystem sorts the accumulated energy vector passed from the Time Windowing subsystem into bins representing frequencies of interest. At this point, elements of the energy vector that are not within the range of a frequency bin are not forwarded for further analysis. The subsystem uses an accumulator to compute the amount of energy in each frequency bin.

a. Timing Analysis

The signals in this subsystem are controlled by two finite state machines. As each element of the energy vector is output from the Time Windowing subsystem, it is stored sequentially in a Dual Port RAM. This process is controlled by the *we_time_win* algorithm. When the last point is written, the algorithm sets the *fft_e* flag. This signals the *re_freq_win* algorithm to read the appropriate Ranges of Interest (ROIs) from memory. State transition diagrams for each algorithm are shown in Figure 18.

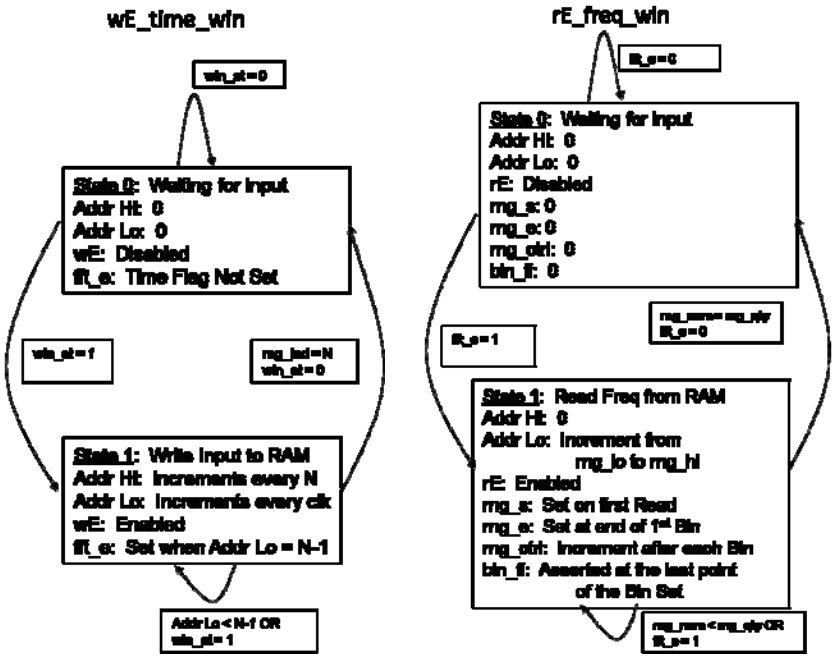

Figure 18. State Transition Diagrams for Frequency Analysis Subsystem.

The *rng_s* and *rng_e* signals align with the start and end of each ROI, respectively. They are generated along with the addresses of the start and end points and then delayed by one clock to align with the start and end points as they are available from memory. These signals are used to control a series of accumulators, which add each sequential input to the value stored on the last clock cycle. The value in the accumulator is reset at the end of each ROI. The *bin_fl* signal is set for one clock cycle at the end of the last ROI. This signal is delayed by two clock signals to align with the data signal as it leaves the accumulators and is renamed *win_fl*.

From this point forward, the timing of the circuit is dependent on the ROI selected and the input signal. To demonstrate the circuit's timing, a test was run with a real input signal, where

$$x[n] = \frac{1}{4}\left(\sin\left(2\pi n \times 3/1024\right) + \sin\left(2\pi n \times 5/1024\right)\right) \text{ for } n = 0...3095. \qquad \text{(IV.5)}$$

As expected from the DFT calculation, the corresponding output from the FFT subsystem is

$$\begin{aligned} \mathrm{Re}(k) &= 0 \forall k \\ \mathrm{Im}(k) &= -0.128 \text{ for } k = 3,5 \\ &= 0.128 \text{ for } k = 1019,1022 \end{aligned} \quad . \qquad \text{(IV.6)}$$

The energy vector output from the Time Windowing Subsystem, with $M = 3$ is

$$E(k) = \begin{cases} 3 \times \left(0^2 + \left(\frac{128}{1024}\right)^2\right) = 0.047 & \text{if } k = 3,5,1019, \text{ or } 1022 \\ 0 & \text{otherwise} \end{cases}. \qquad \text{(IV.7)}$$

The ROIs used for this test were $k = 0...7$ and $k = 1019...1023$. The expected energy in each frequency bin is $2 \times 0.047 = 0.094$. For convenience, $F(x)$ is defined as the total number of FFT points in $\mathrm{ROI}(x)$. In this example there are two ROIs, where $F(0) = 8$ and $F(1) = 5$.

A timing diagram for the test is shown in Figure 19. The first ROI point is read from memory $N+1$ clock cycles after the first energy point is written. In this example, the first energy point is written at $t = 4259$ and the first ROI point is read from memory at $t = 5284$. The sum of energy in each bin is available after every element of the energy vector is entered into memory, the entire ROI has been read, and the signal has left the accumulator. This sum is aligned with the valid signal, which is the same as the *rng_e* signal delayed by two clock cycles. The time each ROI energy sum is available can be expressed as

$$t\left\{\sum ROI\left(y\right)\right\} = \text{ FFT delay} + \overbrace{\left(M-1\right)N+2}^{\text{Time Windowing Delay}} + \overbrace{N+1}^{\text{Write Delay}} + \overbrace{\sum_{x=0}^{y} F(x)}^{\text{Read Delay}}. \qquad \text{(IV.8)}$$

Source	Signal	Sum (Time 1+Time2+Time3)																												
		FFT Period 3																												
		4257	4258	4259	4260	...	5278	5279	5280	5281	5282	5283	5284	5285	5286	5287	5288	5289	5290	5291	5292	5293	5294	5295	5296	5297
Clock	Clock	0	1	2	3	...	1021	1022	1023	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
FFT	p_cnt	1023	0	1	2	...	1020	1021	1022	1023	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FIFO out (k)	1023	0	1	2	...	1020	1021	1022	1023	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Adder In (k)	1023	0	1	2	...	1020	1021	1022	1023	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Adder Out	1023	0	1	2	...	1020	1021	1022	1023	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	pwr_time (k)	0	0	0	1	...	1019	1020	1021	1022	1023	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pwr_time	time_st	0	0	1	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pwr_time	time_end	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pwr_time	time_val	0	0	1	1	...	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pwr_time	win_cnt	1	1	1	1	...	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
we_time	wE	0	0	1	1	...	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
we_time	fft_e	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
re_freq	rng_s	0	0	0	0	...	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
re_freq	rng_e	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
RAM	pwr_time_m	0	0	0	0	...	0	0	0	0	0	0	0	1	2	3	4	5	6	7	1019	1020	1021	1022	1023	0	0
AccumMux	pwr_rng (k)	0	0	0	0	...	0	0	0	0	0	0	0	0	0+1	+2	+3	+4	+5	+6	+7	1019	+1020	+1021	+1022	+1023	
	pwr_rng (val)	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0.047	0.047	0.094	0.094	0.094	0.047	0.047	0.094	0.094	0.094	
re_freq	valid	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	
re_freq	win_fl	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
re_freq	rng_ctl	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	
wind_anal	wE_qty	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

Figure 19. Timing of Frequency Windowing Subsystem.

For the two ROI in this test, ROI(0) and ROI(1), the time that their energy is available is expressed as

$$\begin{aligned} t\left\{\sum ROI\left(0\right)\right\} &= 2209 + 2\times 1024 + 2 + 1024 + 1 + 8 = 5292 \\ t\left\{\sum ROI\left(1\right)\right\} &= 2209 + 2\times 1024 + 2 + 1024 + 1 + 8 + 5 = 5297 \end{aligned} . \quad \text{(IV.9)}$$

This corresponds with the test results displayed in Figure 19. For convenience, the time of the last bin energy calculation in a set will be annotated as $t_{ROI(final)}$.

b. Resource Analysis

Similar to the FIFO buffer in the Time Windowing subsystem, the size of the Dual Port RAM used in this subsystem is dependent on the bit width of the data word and the depth of the memory. As discussed in the Time Windowing section, the output from the adder is in 52_42 format. This format propagates to the input of the Dual Port RAM, setting the bit width for the block at 52. As discussed in [24], the maximum bit width allowable is dependent on the depth of memory and the device.

The depth of the Dual Port RAM block is entered using a fill-in block in the System Generator user interface. For the initial SDR design, this value was 2^{15}. The basis for this was most likely determined by the expression

$$\begin{aligned} \text{Depth} &= N\times(M+1)\times mem\_col \\ \text{Depth} &= 2^{10}\times 2^{2}\times 2^{3} = 2^{15} \end{aligned} . \quad \text{(IV.10)}$$

The corresponding memory requirement is

$$\begin{aligned} \text{Memory} &= \text{Bit Width} \times \text{Depth} \\ \text{Memory} &= 52\times 2^{15} = 1664 \text{ Kbit} = 208 \text{ KB} \end{aligned} . \quad \text{(IV.11)}$$

This configuration of the initial SDR design was only tested at a high level of abstraction in the MATLAB®/Simulink® environment to verify that the algorithm would function as designed. The algorithm cannot run on a device in its current format because this memory requirement far exceeds the capacities of the target devices. The expression in Equation (IV.11) does not represent the minimum memory requirement, so the memory can be used much more efficiently with some adjustments.

Since the Dual-Port RAM only stores one energy vector of length *N* for every *M* FFT periods, the factor $(M+1)$ in Equation (IV.10) can be replaced with (1). The *re_freq_win* has no provision for reading any vector other than the one just written to memory. In a worst-case scenario with continuous, streaming FFT output and $M=1$, the Dual-Port RAM would require enough depth to write the next vector of *N* elements while frequency bins from the last one are being read. Assuming $\sum F(x) \le MN$, the maximum required depth for a 1024-point FFT is

$$\text{Depth} = 2N = 2 \times 2^{10} = 2^{11}. \tag{IV.12}$$

As discussed in [3], this can be accomplished by changing the value in the Dual-Port RAM System Generator interface, setting $mem\_col = 2$ and adjusting the bit width of the *addr_hi* signal to one. The new memory requirement is

$$\begin{aligned} &\text{Memory} = \text{Bit Width} \times \text{Depth} \\ &\text{Memory} = 52 \times 2^{11} = 104 \text{ Kbit} = 13 \text{ KB} \end{aligned} \tag{IV.13}$$

The assumption that $\sum F(x) \le N$ is not necessary for this portion of the circuit. Based on the timing, if $\sum F(x) < NM$, the circuit would still function as long as the memory is modified so that $\text{Depth} \ge NM$. Setting limits on $\sum F(x)$ is necessary to ensure that the circuit works with the minimum required memory. Additionally, the $\sum F(x) \le N$ restriction is required later in the signal flow path. This will be revisited later in the chapter.

D. BIN THRESHOLD ANALYSIS AND DATA MANAGEMENT

The Bin Threshold Analysis subsystem simply compares the total energy in each bin to the user-defined threshold. If the energy in the bin exceeds the threshold, the ROI index is written to a FIFO buffer in the Temporary Data Management subsystem. At the end of the bin set, the window number and the number of bins that passed are also stored in FIFO buffers. These buffers are duplicated in the Header Generation subsystem.

1. Timing Analysis

As each of the bins is analyzed, the *wind_anal* algorithm sets the *wE_rng* signal, which permits the ROI index to be written to memory. At $t_{ROI(final)}$, the algorithm sets the *wE_qty* signal, permitting the window number and number of passed bins to be written to memory. There is no clock delay between the accumulator output from the previous subsystem and memory.

The *wE_qty* also acts as a flag, for the *hdr_data_mgt* algorithm. If the *wE_qty* flag is asserted the algorithm checks to ensure that a previous window is not being read from memory. If the *tmp_busy* flag is not asserted, the algorithm sets the *hdr_fl* flag. After a two-clock delay, this cues the *out_hdr* algorithm to begin generating a header for the downlink data frame.

A state transition diagram for the *out_hdr* algorithm is shown in Figure 20. State Zero outputs the window number as the first part of the header before transitioning to the next state. In State One, the algorithm outputs the number of bins that passed the bin threshold analysis, then transitions to State Two. In State Two, the algorithm outputs the status of the *pri* flag, then transitions to State Three. The length of the final part of the output header is dependent on the number of bins that passed the bin threshold analysis. The algorithm remains in State Three, adding a bin number to the header on each clock cycle, until all bins that passed the bin threshold analysis are added to the header. When all of the appropriate bins have been added to the header, the algorithm sets the *tmp_fl* signal and returns to State Zero. The *tmp_fl* signal cues the *re_tmp* algorithm to read FFT points from temporary memory, sending them to the Output Format subsystem.

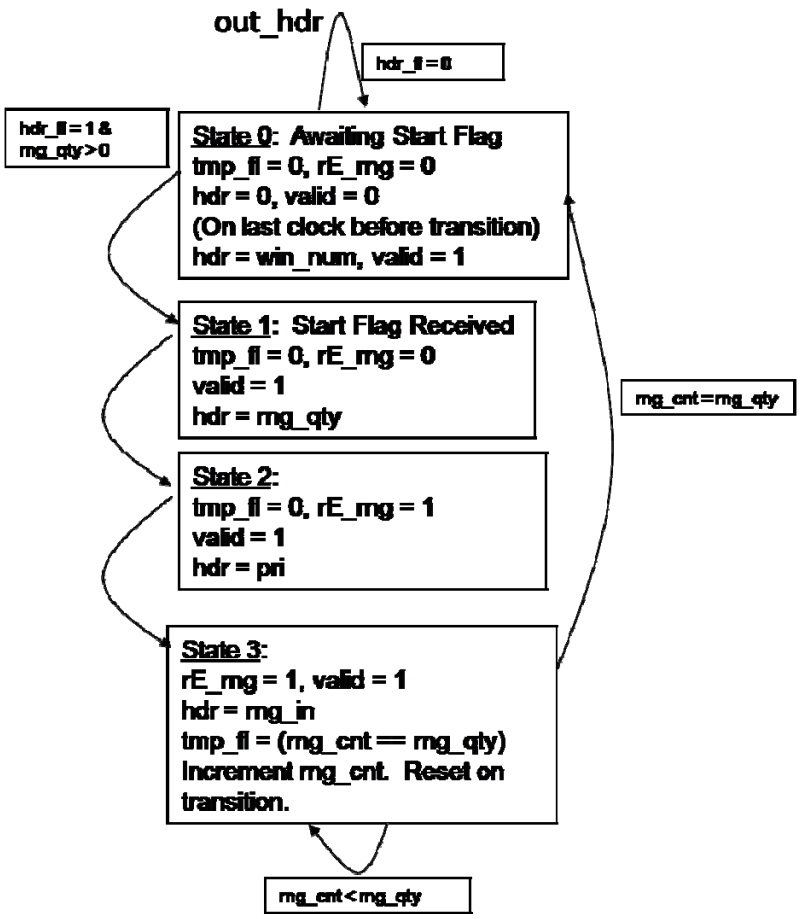

Figure 20. State Transition Diagram for *out_hdr* Algorithm

For convenience, the time the first element of the header is sent to the Output Format subsystem will be annotated as t_{hdr}. The time that the first FFT point is read from temporary memory will be annotated as t_{tmp}. The delay between $t_{ROI(final)}$ and t_{hdr} is two clock cycles. The delay between $t_{ROI(final)}$ and t_{tmp} is expressed as

$$t_{tmp} - t_{ROI(final)} = 2 + \overbrace{3 + \#\,\text{Passed Bins}}^{\text{Header Elements}} + 2\,. \qquad \text{(IV.14)}$$

The additional two-clock delay at the end of the expression indicates the time required to read the ROI from temporary memory then read the first FFT point from temporary memory. The algorithm reads $MF(x)$ points from temporary memory. After each ROI, there is a one-clock delay so the algorithm can read the next ROI from temporary memory.

2. Resource Analysis

The only memory requirements for these subsystems are the two sets of three FIFO buffers used to store ROI information. The initial SDR design sets the depth of these buffers to 16. This sets the maximum number of ROIs that can be passed as 16. The bit width for each value stored is five. The total memory requirement for this subsystem is

$$\begin{aligned} \text{Memory} &= 2 \times 3 \times \text{Depth} \times \text{Bit Width} \\ &= 6 \times 16 \times 5 = 480 \text{ bits} = 60 \text{ Bytes} \end{aligned} \tag{IV.15}$$

This amount of memory is very small compared with the total memory available and the other memory requirements of the circuit. It is not considered in approximations.

The *re_tmp* algorithm finds the values in temporary memory by using the current relative time window as an index, where each indexed time window has *MN* elements. The *re_tmp* algorithm determines the current time window index from the *win_num* signal, which is incremented after every bin set by the *wind_anal* algorithm. The signal is reset when $win\_num == mem\_col - 1$. Although this has no impact on the memory requirements of this subsystem, this sets a key constraint on the amount of temporary memory required. The memory depth must be an integer multiple of *MN*.

E. TEMPORARY STORAGE AND OUTPUT CONTROL

1. Temporary Storage Subsystem

The two Dual-Port RAM blocks in this subsystem constitute the largest memory requirement for the overall circuit. The memory must be able to store *NM* points while the circuit determines which points to downlink. Since the temporary memory must continue storing additional FFT points while the energy is being calculated, additional space is required. This amount is dependent on the amount of time required to compute the energy and downlink points of interest. After all points of interest from a given time window have been read, the memory that stored the time window can be overwritten.

As discussed in previous sections, the timing of the circuit's output is dependent on $\sum F(x)$ and the number of bins that pass the bin threshold analysis. By setting a restriction that $\sum F(x) \le N$, a worst-case scenario can be examined where $\sum F(x) \approx N$ and every bin passes the bin threshold analysis. Any additional overhead can be neglected, provided $N \gg 16$. In this approximation, the minimum memory depth required is

$$\begin{aligned}\text{Temp Memory Depth} &= \overbrace{(M-1)N}^{\text{Time Window}} + \overbrace{N + \sum_{x=0}^{\#ROI-1} F(x)}^{\text{Divide into bins}} + \underbrace{M\sum_{x=0}^{\#ROI-1} F(x)}_{\text{Read passed bins from memory}} \\ &= MN + N + MN = N(2M+1)\end{aligned} \quad . \quad \text{(IV.16)}$$

It is important to note here that the $\sum F(x) \le N$ restriction is necessary for the circuit to function. If $\sum F(x) > N$ and every bin passes the bin threshold analysis, then it would take longer to read values out of temporary memory than it would to write them. If this behavior persisted over several time windows, the circuit would eventually run out of memory regardless of the depth. If this functionality is desired, the circuit would need to be modified to ensure that FFT points in overlap regions between bins are not read more than once.

As stated in the previous section, the temporary memory must be addressable in integer multiples of *MN*. This can be done either by either rounding up to $3MN$ or rounding down to $2MN$. In order to round down, the ROI size must be further restricted such that

$$\begin{aligned} MN + \sum_{x=0}^{\#ROI-1} F(x) + M\sum_{x=0}^{\#ROI-1} F(x) &\le 2MN \\ \sum_{x=0}^{\#ROI-1} F(x) &\le \frac{MN}{(M+1)} \end{aligned} \quad . \quad \text{(IV.17)}$$

In an example where $M = 3$ and $N = 1024$, the ROI size would be restricted to

$$\sum_{x=0}^{\#ROI-1} F(x) \le \frac{3(1024)}{(3+1)} = 768 \quad \text{(IV.18)}$$

The restriction in Equation (IV.17) indicates the minimum memory requirement for this subsystem. As can be inferred from the inequality, in order to achieve a memory requirement of *MN*, the total ROI size must equal zero. As *M* increases, a larger fraction of *N* points can be included in the ROI because the fraction of time associated with overhead becomes smaller.

Another consideration when determining Temporary Memory size is the fact that the memory depth of the dual-port RAM must correspond to the bit width of the signal used to address memory. If *w* is the bit width of the memory address, then the depth of memory must be 2^w. If *M* is not a power of two, portions of Temporary Memory will be assigned but left unused.

The initial SDR design set a Temporary Memory depth of 32*N* for each Dual-Port RAM, which restricts the number of FFT periods in each time window to $M \leq 15$. As discussed in previous sections, the bit width of the FFT data points is 35 in this design. The resulting memory used is

$$\begin{aligned} \text{Memory} &= 2 \times \text{Depth} \times \text{Bit Width} \\ &= 2 \times 2^5 \times 2^{10} \times 35 = 2240 \text{ Kbit} = 280 \text{ KB} \end{aligned} . \qquad \text{(IV.19)}$$

2. Output Format Subsystem

The performance of this subsystem is discussed in [3]. There is no delay between the time an output signal enters the subsystem and the time that it is written to the downlink FIFO buffer. This subsystem manages the input to the downlink FIFO buffer using signals produced by previous subsystems.

The only memory requirement for the Output Control Subsystem is the FIFO buffer that holds the output data frame while waiting for an external downlink signal. In the initial design testing, downlink was only disabled for short periods of time to verify that the circuit would restrict the amount of data chosen for downlink when the buffer reached 25% of its capacity. For this reason, the FIFO buffer depth is only 32. For

maximum downlink flexibility, the end user should use all remaining memory resources to increase the size of this buffer. The amount of memory used in the initial SDR design is

$$\begin{aligned}\text{Memory} &= \text{Depth} \times \text{Bit Width} \\ &= 32 \times 70 = 2240 \text{ bits} = 280 \text{ Bytes}\end{aligned}. \tag{IV.20}$$

F. GENERALIZED CIRCUIT EXPECTATIONS

The previous sections discussed the timing and resource usage of each subsystem associated with the portions of the initial SDR design used for signal compression. The following generalizations summarize the conclusions reached in the previous sections. These equations were created based on both analysis of the circuit's design and observations from simulations where $N = 1024$ and $M = 3$. The expressions should hold valid for any configuration where $N \geq 16$ and M is a power of two.

The delay from the first signal input to the first output indicates the overall latency of the design. The delay is not calculated after the output values are written to the downlink FIFO buffer because the rate at which values are read out is dependent on conditions external to the circuit. The delay to the time that the first header element is written to the downlink FIFO buffer (t_{hdr}) is expected to be

$$t_{hdr} = \text{FFT Latency} + \overbrace{N(M-1)+2}^{\text{Time Window}} + \overbrace{N+1+\sum_{x=0}^{\#\text{ROI}-1} F(x)+1+2}^{\text{Frequency Window}}. \tag{IV.21}$$

The delay from the first signal input to the time that the last output is written to the downlink FIFO buffer indicates the amount of time required to generate the data frame for downlink (t_{frame}). If the number of passed bins is expressed as P, then t_{frame} can be expressed as

$$t_{frame} = t_{hdr} + \overbrace{3+P}^{\text{Frame Header}} + 2 + \overbrace{M\sum_{x=0}^{P} F(x)}^{\text{Transmit Passed Bins}} + \overbrace{P-1}^{\text{Delay Between Successive Bins}}. \tag{IV.22}$$

By approximating the total ROI size as $\sum F(x) \approx MN/(M+1)$, the total amount of memory required for the system can be expressed as

$$\text{Memory} \cong \text{FFT Memory} + \overbrace{(76 \text{ bits})N}^{\text{Time Window FIFO}} + \overbrace{(54 \text{ bits})2N}^{\text{Dual-Port RAM}} + \overbrace{(35 \text{ bits})4MN}^{\text{Temp Memory}} + \text{Pre-Downlink Storage} \quad \text{(IV.23)}$$

G. SUMMARY

This chapter provided a detailed analysis of the initial SDR design. State transition diagrams were created to further illustrate the function of the design's control algorithms. General circuit equations were developed to create expectations for the circuit's timing and memory resource requirements. These expressions can be used as design equations to determine appropriate values for *M*, *N*, and ROI size based on the desired performance and the resources available in the target device. The next chapter explains how this information can be used to make changes that improve both the efficiency and the functionality of the design.

IV. INITIAL MODIFICATIONS TO THE ORIGINAL DESIGN

The conclusions reached in the previous chapter facilitate making changes to the circuit without impact to its overall functionality. This chapter examines how the circuit can be adjusted to improve its portability between different FPGA devices. It also introduces bandwidth and resource-conserving measures through the use of the conjugate symmetry property inherent in Fourier Transforms of real signals. This chapter demonstrates the use of the FFTv1.0 IP with the SDR circuit, permitting the circuit's use on a Virtex™-I FPGA.

A. INCREASE COMPRESSION AND MEMORY EFFICIENCY

This section discusses how the conjugate symmetry property for Fourier Transforms of real signals can be used to improve the storage and downlink efficiencies of this circuit.

1. Theory

As discussed in [9], if the input signal $x(t)$ to a Fourier Transform is real then the output $X(F)$ is conjugate symmetric as shown in the expression

$$X(-F) = X^*(F), \text{ where } x(t) = x^*(t). \tag{V.1}$$

Similarly, the Discrete Fourier Transform has a symmetric property. For a real input sequence of $x[n], n = 0,..., N-1$, the corresponding output is

$$X[k] = X^*[N-k] \text{ for } k = 0,..., N-1. \tag{V.2}$$

From this expression, it is evident that if the first $N/2$ points of the FFT output are known, the remaining output points can be reproduced. Therefore, if the first $N/2$ FFT output points are used for analysis, storage, and downlink selection then the remaining $N/2$ FFT output points are redundant. This information can be used to improve the way the SDR circuit stores and compresses information.

2. Changes Made

The FFT IP is implemented in a black box, so this portion of the circuit cannot be changed. All portions of the circuit downstream from the FFT IP need to be adjusted to use only half of the output points. The most dramatic change is to the Time Windowing subsystem, which controls the timing of all further processing. The Frequency Windowing subsystem and Temporary Memory subsystem were adjusted to accommodate reductions in the amount of memory required. All other portions of the circuit did not require any changes because their functions are not directly dependent on the number of points being processed.

a. Changes to the Time Windowing Subsystem

As discussed in Chapter III, the Time Windowing subsystem is controlled by the *pwr_time* algorithm. In the initial SDR design, the *pwr_time* algorithm worked on the assumption that on each clock cycle a new point was entering the subsystem to be processed. As shown in Figure 16, if the last set of FFT points within a time window was received and the *prep* flag was not set then the algorithm would return to a waiting state, resetting all internal variables.

This algorithm was streamlined to only calculate the sum of the energy in the first $N/2$ points. An additional waiting state was added, identified as State Two and displayed in the modified state transition diagram shown in Figure 21. State Two is entered after the first $N/2$ points are written to the FIFO buffer. This state retains all internal variables, keeping track of the number of FFT output sets that have been processed. On cue from the *prep* flag, the algorithm transitions from State Two to State Three if the current FFT set is not the last one in the time window. If the current FFT set is the last one in the time window, the algorithm transitions from State Two to State Five. Similar to State Three in the original design, State Five activates output flags and disables writing to the FIFO.

Two additional states were added to reflect the fact that the circuit behaves differently for two clock cycles following the last FFT input to be processed within a set.

State Four and State Six immediately follow State Three and State Five, respectively. State Four ensures that writing to the FIFO is enabled for two clock cycles. State Six ensures that the output is enabled. Adding these states brings the algorithm closer to the Moore machine FSM model, although the output of State Three is still dependent on an internal delay flag. State Seven was added to permit streamlined processing in the case where $M = 1$. Instead of storing FFT points in the FIFO buffer, they are routed directly to the subsystem output, and output flags are enabled.

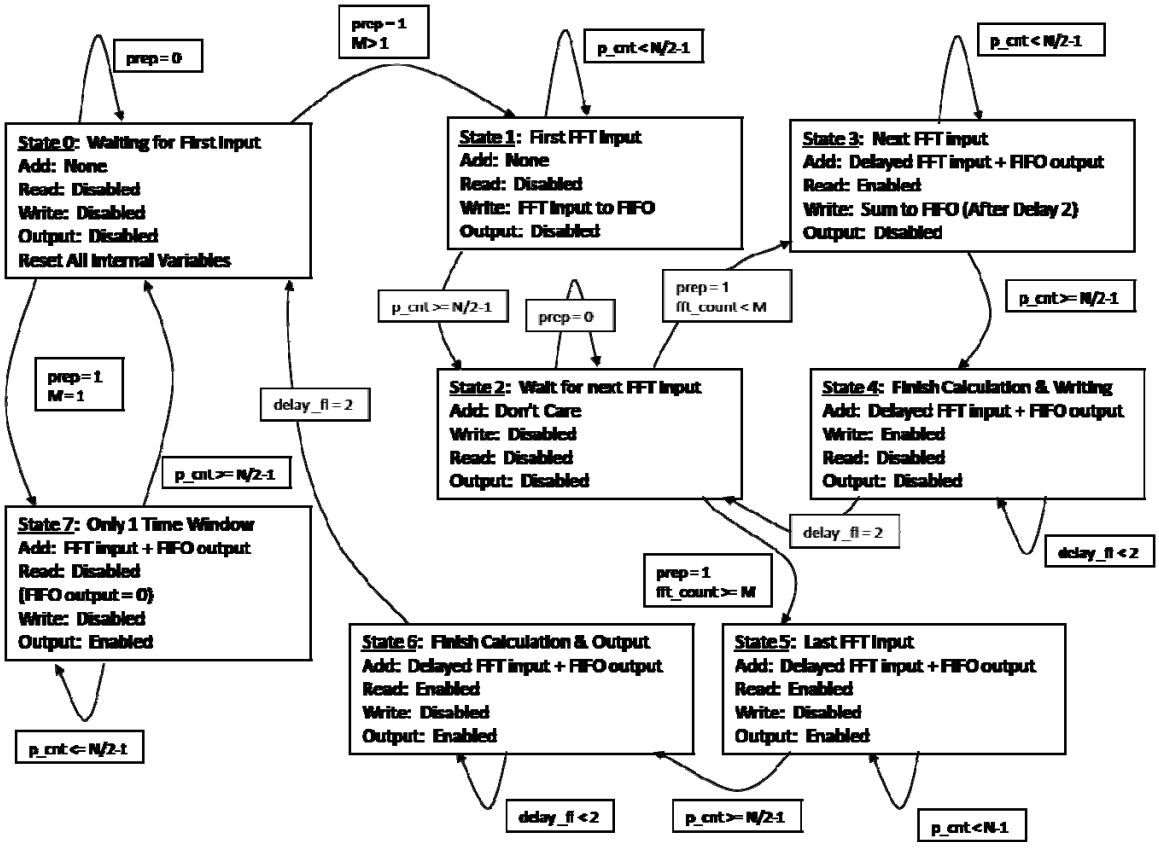

Figure 21. State Transition Diagram for the Modified *pwr_time* Algorithm.

The algorithm must still wait for the FFT to process $N(M-1)$ points before it can begin outputting the summed energy vector. As a result, there is no difference in the amount of time required to calculate the energy in the time window. An updated timing chart, reflecting the new states is shown in Figure 22.

Source	Signal							Time Window 1								Sum (Time 1+Time2) / Time Window 2										
		2208	2209	2210	2211	...	2718	2719	2720	2721	2722	...	3231	3232	3233	3234	3235	...	3742	3743	3744	3745	3746	3747	...
FFT	p_cnt		0	1	2	...	509	510	511			...			0	1	2	...	509	510	511				...
FFT	prep	1				...						...		1											...
Internal	state	**0**	**1**	**1**	**1**	**...**	**1**	**1**	**1**	**2**	**2**	...	**2**	**2**	**3**	**3**	**3**	**...**	**3**	**3**	**3**	**4**	**4**	**2**	...
pwr_time	we	0	1	1	1	...	1	1	1	0	0	...	0	0	0	0	1	...	1	1	1	1	1	0	...
pwr_time	re	0	0	0	0	...	0	0	0	0	0	...	0	0	1	1	1	...	1	1	1	0	0	0	...
pwr_time	mem_mux	FFT	FFT	FFT	FFT	...	FFT	FFT	FFT	X	X	...	X	X	FFT	FFT	Add	...	Add	Add	Add	Add	Add	X	...
pwr_time	add_mux	FFT	FFT	FFT	FFT	...	FFT	FFT	FFT	X	X	...	X	X	0	Del	Del	...	Del	Del	Del	Del	X	X	...
	FIFO in	X	0	1	2	...	509	510	511			...			X	X	1	...	508	509	510	511	512		...
	FIFO out	0	0	0	0	...	0	0	0			...			0	1	2	...	509	510	511	512			...
	Adder In (k)		0	1	2	...	509	510	511	X	X	...	X	X	0	1	2	...	509	510	511	512	X	X	...
L = 0	Adder Out		0	1	2	...	509	510	511	X	X	...	X	X	0	1	2	...	509	510	511	512	X	X	...
	pwr_time (k)	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	0	...
pwr_time	time_st	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	0	...
pwr_time	time_end	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	0	...
pwr_time	time_val	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	...	0	0	0	0	0	0	...
pwr_time	win_cnt	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1	1	1	1	1	1	1	1	1	...

Source	Signal			Sum (Time 1+Time2+Time3) / Time Window 3									
		4255	4256	4257	4258	4259	4260	...	4766	4767	4768	4769	4770	4771
FFT	p_cnt			0	1	2	3	...	509	510	511			
FFT	prep		1					...						
Internal	state	**2**	**2**	**5**	**5**	**5**	**5**	**...**	**5**	**5**	**5**	**6**	**6**	**0**
pwr_time	we	0	0	0	0	0	0	...	0	0	0	0	0	0
pwr_time	re	0	0	1	1	1	1	...	1	1	1	1	0	0
pwr_time	mem_mux	X	X	X	X	X	X	...	X	X	X	X	X	X
pwr_time	add_mux	X	X	Del	Del	Del	Del	...	Del	Del	Del	Del	Del	X
	FIFO in					X	X	...	X	X	X			
	FIFO out				1	2	3	...	509	510	511	512	0	0
	Adder In (k)	X	X	X	1	2	3	...	509	510	511	512	0	0
L = 0	Adder Out	X	X	X	1	2	3	...	509	510	511	512	0	0
	pwr_time (k)	0	0	0	0	1	2	...	508	509	510	511	512	0
pwr_time	time_st	0	0	0	0	1	0	...	0	0	0	0	0	0
pwr_time	time_end	0	0	0	0	0	0	...	0	0	0	0	1	0
pwr_time	time_val	0	0	0	0	1	1	...	1	1	1	1	1	0
pwr_time	win_cnt	1	1	1	1	1	1	...	1	1	1	1	1	2

Figure 22. Timing of Modified *pwr_time* Algorithm.

b. Changes to Frequency Windowing Subsystem

The Frequency Windowing subsystem required only a few minor changes to ensure that it makes the most efficient use of memory based on the new output from the Time Windowing subsystem. By modifying the restriction on ROI size so that $\sum F(x) < N/2 - 1$, the algorithm will have time to read all stored energy points before the next energy vector needs to be written to memory. In a worst-case scenario, where $M = 1$, all points must be written to memory and read out by frequency bin within N clock cycles. This ensures that the next time window energy vector will not overwrite memory that has not yet been read. The *we_time_win* algorithm was adjusted to write only $N/2$ energy points to memory. The first point is read from memory after a write delay of $N/2+1$. The time that the last read occurs should be at least one clock before the next energy vector is written to memory. By restricting the ROI size so that $\sum F(x) = N/2 - 2$, this constraint will be met. If the user orders the frequency bins in such a way that the first points of the energy vector are read first, this restriction could be eased since there is less risk of memory overwrite.

Adding these restrictions on circuit timing and ROI size ensured that the Frequency Windowing subsystem could be implemented using a RAM depth of only $N/2$. This eliminated the need for the *addr_hi* signal, which was used to permit the storage of multiple time window energy vectors. Since only one time window energy vector is stored at a time, the *mem_col* signal used to control how many energy vectors are written to memory was made obsolete and removed as an input to the subsystem. The *addr_lo* signal was adjusted to use only nine bits for FFT index addressing.

c. Changes to Temporary Memory

As discussed in Chapter III, the required temporary memory depth is dependent on the amount of time required to compute the energy and downlink points of interest. In the initial configuration, the temporary memory must be addressable in integer multiples of *MN*. In an optimal memory configuration of *2MN*, the total ROI size must be restricted as shown in Equation (IV.17).

As discussed in the previous section, in the modified circuit the ROI is restricted so that $\sum F(x) < N/2 - 1$. This easily meets the criteria of Equation (IV.17) to use the optimal memory configuration of *2MN*. Since the modified circuit only needs to store the first $N/2$ points, the memory requirement is reduced to $2M\left(N/2\right) = MN$.

While the original circuit needed to contend with continuous streaming input, the modified circuit has a delay of $N/2$ clock cycles before successive time windows are written to memory. If the ROI size were restricted so that all of the information could be transmitted within $N/2$ clock cycles, then the memory depth could be limited to $MN/2$. In order for this to occur, the ROI must be restricted to satisfy the inequality

$$\sum F(x) + M\sum F(x) < N/2$$
$$\sum F(x) < \frac{N}{2\left(M+1\right)} \quad . \tag{V.3}$$

Unlike the inequality of Equation (IV.17), increasing *M* further restricts the permissible size of the ROI. This option is explored as a possibility for further reducing memory requirements, but is not recommended for circuits using continuous FFT output since it could severely restrict the circuit's utility to the end user.

The *we_tmp* algorithm was adjusted to return to a waiting state after $N/2$ points are written to temporary memory, preserving an internal variable to keep track of how many FFT periods have been stored. The algorithm still relies on the *mem_col* signal to determine how many FFT periods should be stored in memory. The algorithm resets its high-level memory address output to zero when the number of stored FFT periods is equal to $mem\_col \times M$. The *re_tmp* algorithm was adjusted to ensure that the bit widths of the memory addresses matched the new size of the RAM module in the Temporary Memory subsystem. No other changes were necessary because the *re_tmp* algorithm receives its cues from other algorithms already adjusted for a $N/2$ configuration.

d. Changes to Test Configuration

The examples shown in Chapter III used a ROI with $k = 1019...1023$. Since these points are now excluded from storage and analysis new ROIs were created to test the performance of the circuit. Modifications to the design were tested using a configuration with $N = 1024$, $M = 3$, and user-defined ROIs $k = 0...5$ and $k = 6...9$. The input signal was also adjusted to ensure energy was available in each frequency bin. Additional frequencies were added to the signal described in Equation (III.5) to ensure the FFT would produce detectable output for $k = \{3, 5, 7, 9\}$.

3. Update to Circuit Generalizations

The changes made to the circuit necessitate updates to the timing and memory expressions. Equation (IV.21) expresses the delay to the time that the first header element is written to the downlink FIFO buffer (t_{hdr}). In the modified circuit, this value is expressed as

$$t_{hdr} = \text{FFT Latency} + \overbrace{N(M-1)+2}^{\text{Time Window}} + \overbrace{\frac{N}{2}+1+\sum_{x=0}^{\#\text{ROI}-1} F(x)+1}^{\text{Frequency Window}} + 2 \qquad \text{(V.4)}$$

There is no change to Equation (IV.22), which includes t_{hdr} in the expression. Equation (IV.23) expresses the amount of memory required for the system. In the modified circuit, this value is expressed as

$$\text{Memory} \cong \text{FFT Memory} + \overbrace{(76 \text{ bits})\frac{N}{2}}^{\text{Time Window FIFO}} + \overbrace{(52 \text{ bits})\frac{N}{2}}^{\text{Dual-Port RAM}} + \overbrace{(35 \text{ bits})2MN}^{\text{Temp Memory}} + \text{Pre-Downlink Storage} \qquad \text{(V.5)}$$

B. INTEGRATING NEW FFT IP

As discussed in Chapter III, the initial SDR design used the FFTv4.1 IP to compute the FFT. Chapter II highlighted some of the differences between the FFTv4.1 IP and the FFTv1.0 IP. If a Virtex™-1 FPGA is the desired target device, the circuit cannot use the FFTv4.1 IP. This section addresses how the circuit must be modified to accommodate the FFTv1.0 IP, which is compatible with a Virtex™-1 FPGA.

1. Replacing FFTv4.1

The FFT configuration used in the initial SDR design is shown in Figure 3. The updated configuration is shown in Figure 23. As discussed in Chapter II, the original design manually scaled the output by $1/N$. The FFTv1.0 automatically scales the output by this factor, so these scaling blocks were removed. The *start* signal is connected to the *vin* FFTv1.0 input, indicating when the FFT should begin computing. The real and imaginary input signals are connected to the appropriate input ports. The imaginary portion of the input signal is set to zero outside the subsystem.

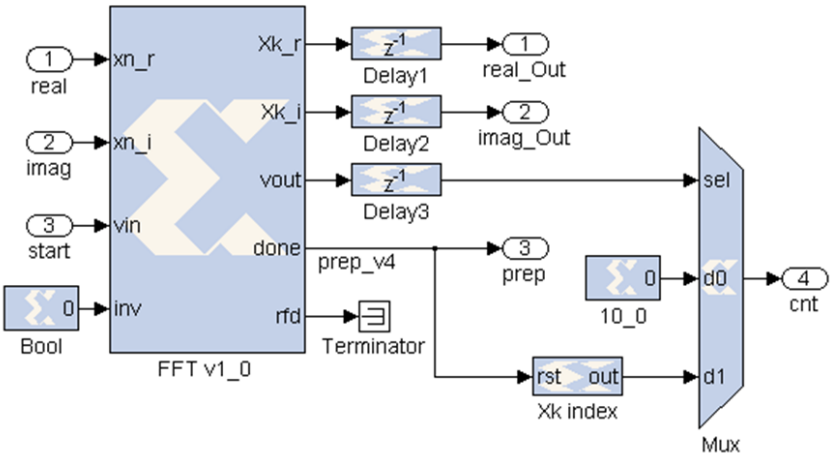

Figure 23. FFTv1.0 as Used in the SDR.

Since the FFTv1.0 IP does not represent the *e_done* signal in its System Generator interface, the signal is simulated from the *done* signal by delaying all other output signals one clock period. The *Xk_index* signal is also unavailable through the System Generator interface. This signal is simulated using a counter. The counter is reset by the *done* signal and counts from zero to 1024, incrementing each clock period. The *valid* signal is used to control a multiplexer. This ensures that the counter output is forwarded to the next subsystem if the FFT output is valid. A constant zero is forwarded if the data is not valid.

2. Implementation Issues

As discussed in Chapter II, the FFTv1.0 samples incoming data points at one quarter of the clock rate used for processing. In the System Generator environment, this is controlled by setting the Simulink® system period to 0.25 in the System Generator module interface as discussed in [3] and shown in Figure 24. Adjusting the sampling rate for the FFTv1.0 IP module changes the sampling rates of other modules within the design. This causes problems with the finite state machines in the remainder of the circuit, which must sample input at the same frequency as the system clock rate.

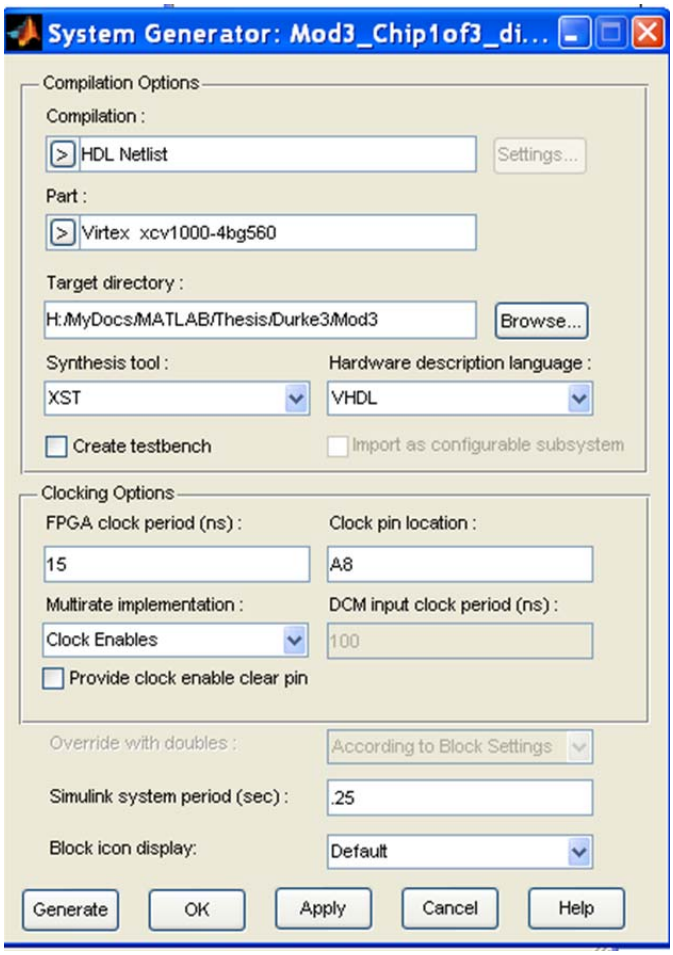

Figure 24. System Generator Configuration for FFTv1.0 [After 16].

The workaround is to divide the circuit into two separate Simulink® designs. From a development standpoint, this means that the circuit must be compiled in separate

parts. The two Simulink® designs can be used to generate HDL netlists that can be re-integrated in the ISE environment. If more than one FPGA is available in the final circuit configuration, it may be desirable to run the FFT computation and the remainder of the SDR circuit on separate FPGAs.

The first Simulink® model for the adjusted circuit is shown in Figure 25. The second Simulink® model is shown in Figure 26. When testing in the Simulink® environment, the FFT model is run first. The output of the first simulation is stored in the MATLAB® workspace variable *FFTout* as a two-dimensional matrix with four columns. This information is separated and reformatted using M-Code so that it can be used as input for the compression algorithm. Appendix A discusses the execution of tests in the Simulink® environment in further detail.

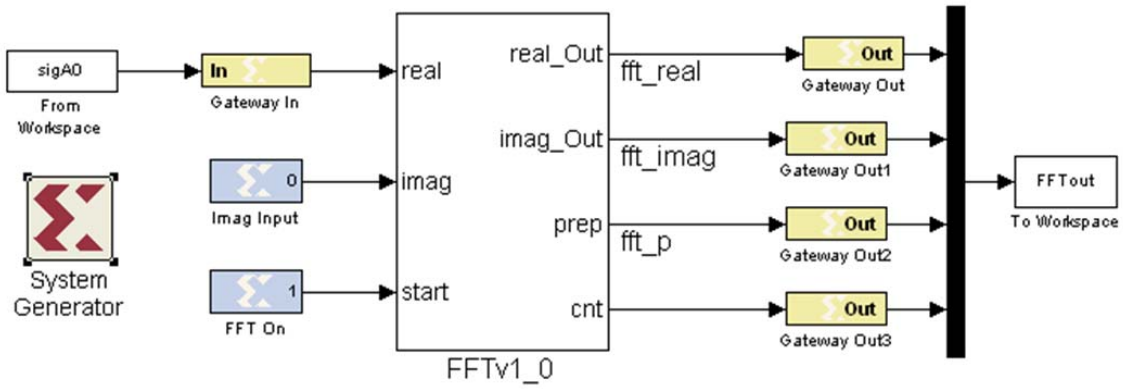

Figure 25. FFTv1.0 separated from Compression Algorithm.

3. Changes to Performance Expectations

As discussed in Chapter II, there is a $3N$ clock delay between the last output of one FFT period and the first output of the next FFT period. The new *pwr_time* algorithm handles this gracefully because it already anticipates a delay between valid inputs. The same is true for the *we_tmp* algorithm. The expression for t_{hdr} is updated for the FFTv1.0 IP to show that

$$t_{hdr} = \text{FFT Latency} + \overbrace{4N(M-1)+2}^{\text{Time Window}} + \overbrace{\frac{N}{2}+1+\sum_{x=0}^{\#ROI-1} F(x)+1}^{\text{Frequency Window}} + 2. \qquad \text{(V.6)}$$

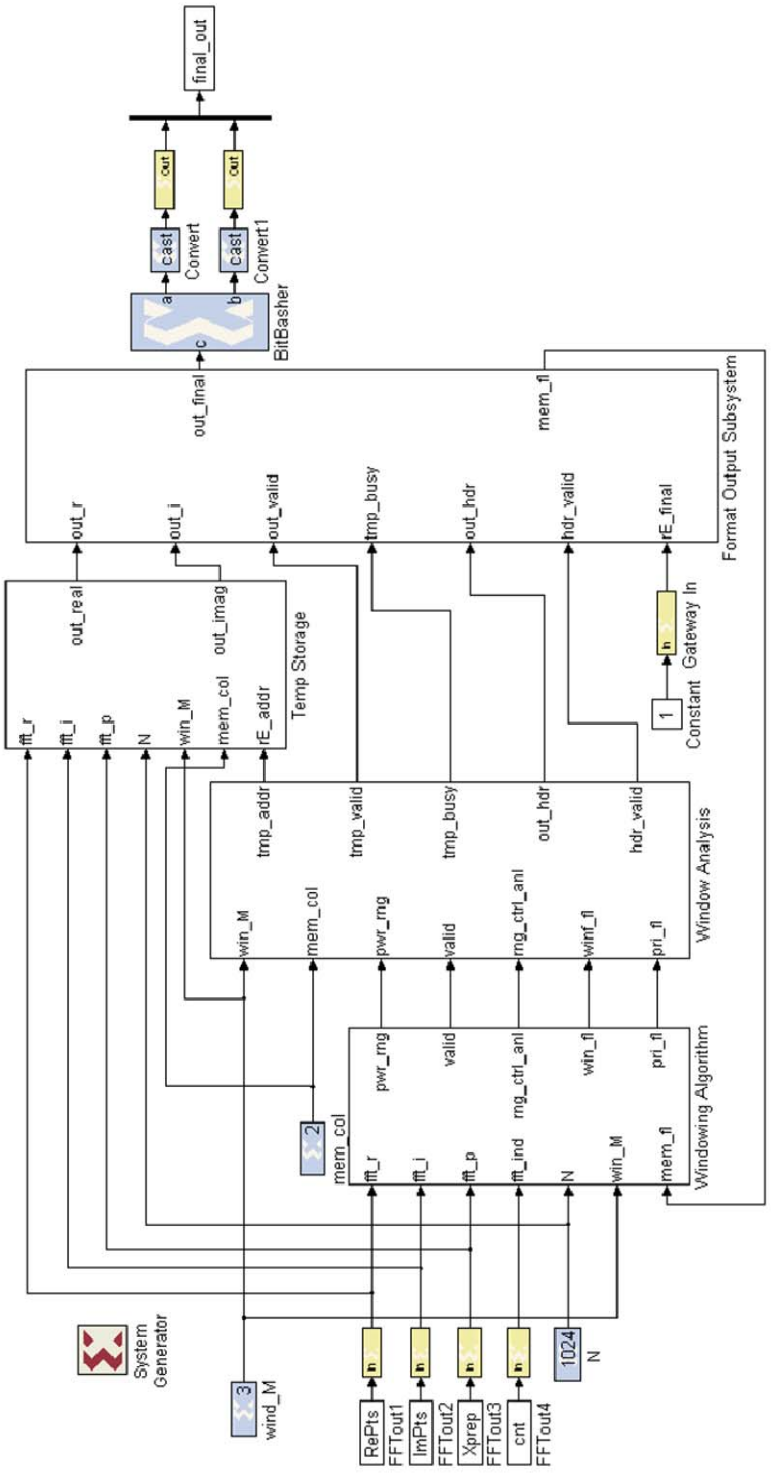

Figure 26. Compression Algorithm Separated from FFT [After 3].

The *3N* clock delay means that the restriction on ROI size $\sum F(x)$ shown in Equation (V.3) can be relaxed. Equation (V.3) indicates that all information needs to be transmitted within $N/2$ clock cycles after the last FFT output point is written to temporary memory. The additional delay means that *3N* can be added to the right side of the inequality so that

$$\sum F(x) + M\sum F(x) < 3.5N$$
$$\sum F(x) < \frac{3.5N}{(M+1)} \quad . \tag{V.7}$$

The adjustment to the inequality means that holding only one time window in memory is now a feasible option, provided $M \le 6$. As discussed in Chapter II, the output bit width of the FFTv1.0 IP is less than the output bit width of the FFTv4.0 IP. This introduces the potential for additional memory savings in a circuit that uses the FFTv1.0 IP. The reduced memory requirement can be expressed as

$$\text{Memory} \cong \text{FFT Memory} + \overbrace{(32\text{ bits})\frac{N}{2}}^{\text{Time Window FIFO}} + \overbrace{(32\text{ bits})\frac{N}{2}}^{\text{Dual-Port RAM}} + \overbrace{(16\text{ bits})2M\,\frac{N}{2}}^{\text{Temp Memory}} + \text{Pre-Downlink Storage}. \tag{V.8}$$

C. ADJUSTING HEADER FORMAT AND DOWNLINK CONTROL

This section discusses changes that improve the efficiency of the initial SDR design's output mechanisms. Changes were made to the header format, and a throttling mechanism was added to ensure that only valid data is sent to the output.

1. Changes to the Header Format

As discussed in Chapter III, the initial SDR design uses a variable-length header to indicate the time window being transmitted, the number of ROI that passed the bin analysis, whether the circuit was operating in a constrained memory condition, and which ROI passed the bin analysis. Each element of the header is transmitted on successive clock periods. This format is shown in Figure 27.

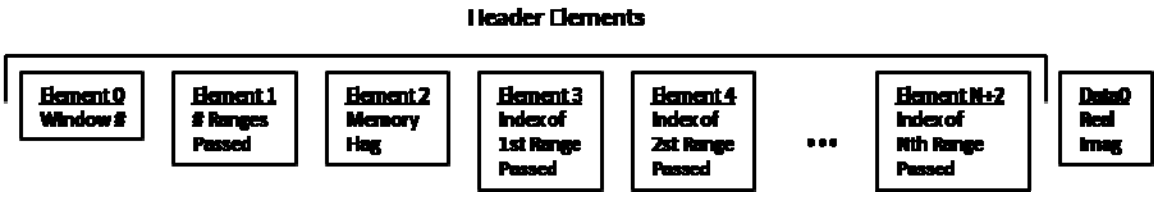

Figure 27. Initial SDR Design Header Format.

As discussed in Chapter III, the initial SDR design transmits a 70-bit data word on each clock cycle. The format of the header in the initial design is extremely inefficient in its use of downlink bandwidth. The information in the header can be compressed into a more efficient fixed-length format. One way to ensure the header has a fixed length is to restrict the number of ROI that can be used. Setting the possible number of ROI to a predetermined value means that only one bit is required for each ROI to indicate whether it passed bin analysis or not. This method also removes the need to transmit the number of ranges that passed bin analysis as a separate quantity since this could be computed by summing the ROI that were flagged. A modified header format is shown in Figure 28.

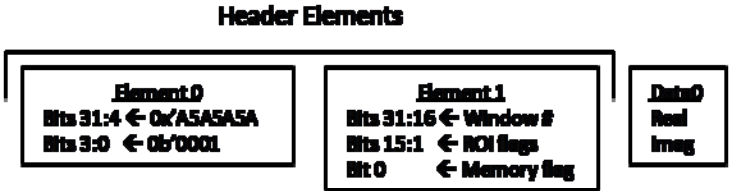

Figure 28. Modified SDR Header Format.

The size of the header was reduced to accommodate an output bit width of 32, which is the minimum amount required to transmit the output of the FFTv1.0 IP. If the output bit width is larger, zeroes can be prepended to the header to match the output data format. The format of the header was changed to pass all required information within two header elements. The first element provides room for expansion, should additional information be required at a later time. In its current format the first element contains a 28-bit preamble for synchronization, followed by 16 bits used for version control. The second element uses 16 bits to indicate the window number. Although this is a small number for all examples used with this thesis, this slot could be used in the future to

communicate the time that the first FFT input was sampled for the time window. The next 15 bits are flags to indicate whether the user-defined ROI passed bin analysis. The last bit indicates if the circuit is operating under a memory constrained condition.

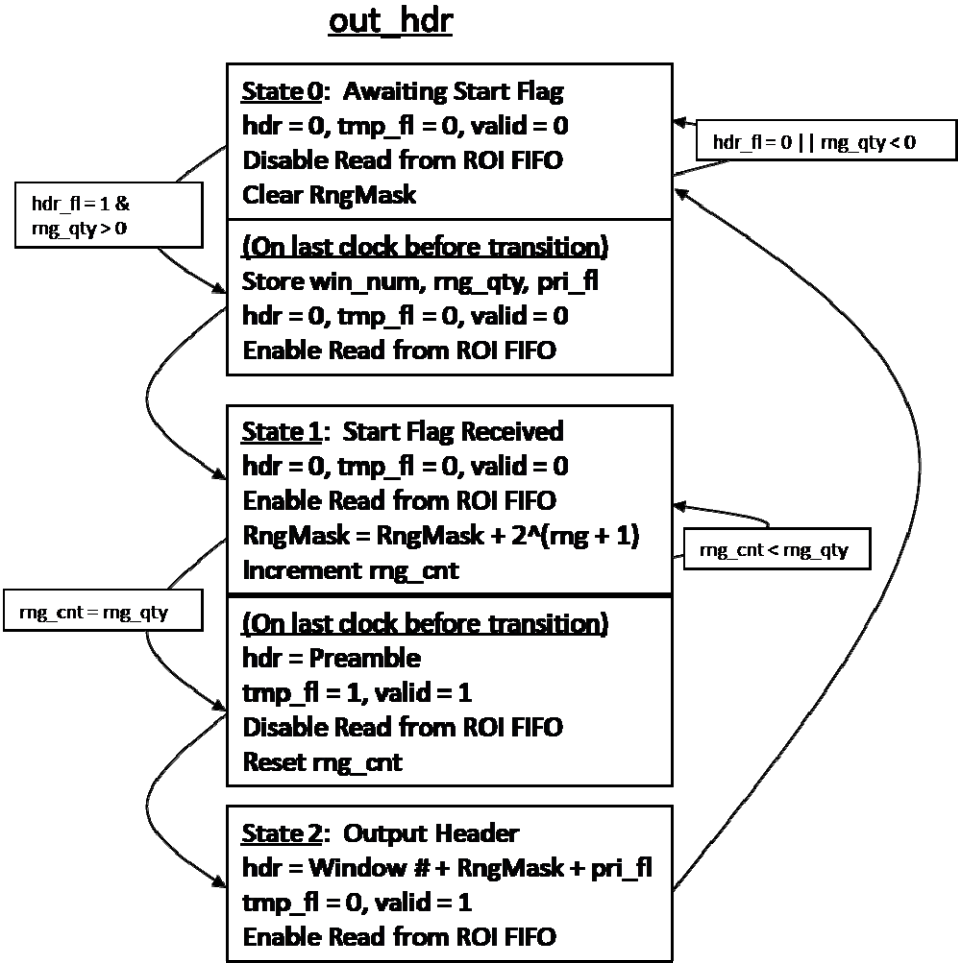

Figure 29. Modified State Transition Diagram for *out_hdr* Algorithm.

These changes were implemented by modifying the *out_hdr* algorithm. The new state transition diagram is shown in Figure 29. In State Zero, the algorithm waits for a flag to indicate that all information is available to generate the header. The algorithm receives the start flag, window number, range quantity, and memory flag at the same time and stores the values in internal variables. As a passing ROI is read from a FIFO buffer, the value is evaluated using a switch statement. The switch statement sets a bit in a 16-bit temporary mask corresponding to the appropriate ROI. The temporary mask is then

added to an ROI mask that is preserved until the header is ready to downlink. This addition could be implemented using a bitwise OR. As soon as all ROI have been processed, the algorithm outputs the first header element. On the next clock cycle, the algorithm outputs the window number, ROI mask, and memory flag. Finally, the algorithm resets all internal variables and transitions back to the waiting state.

2. Changes to Downlink Control

As discussed in Chapter III, the initial SDR design used an external read enable signal called *rE_final* to control when information is transmitted from the downlink FIFO buffer. While this seems like a practical means of leaving downlink decisions in the hands of an external communications system, the initial design did not provide a means of signaling to the external system when information was available to be transmitted. Additionally, there is a one clock cycle delay between each ROI that is output from temporary memory. If this delay is forwarded directly to the transmitted output then for each transmitted ROI, one clock cycle is wasted transmitting invalid information.

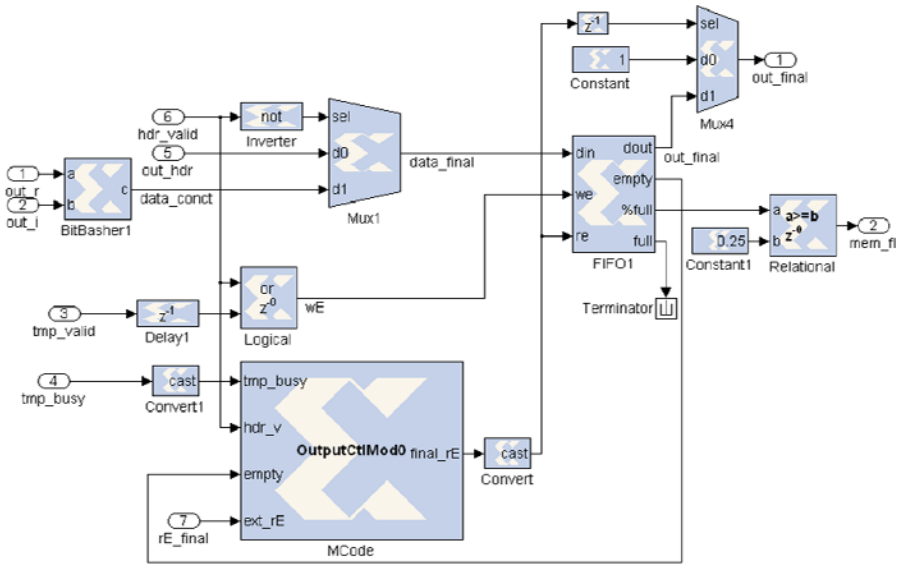

Figure 30. Modified Format Output Subsystem [After 3].

Rather than simply forward a *data_valid* signal to the output and leave decisions in the hands of an external circuit, internal measures were added to enforce greater

control over the SDR output. Changes made to the Format Output subsystem are shown in Figure 30. A multiplexer was added to transmit a constant one if the output data is invalid. A finite state machine called *OuputCtl* was implemented in M-Code to control the *re* input signal for the downlink FIFO buffer based availability of valid information and the external system's ability to receive the information. The state transition diagram for this algorithm is shown in Figure 31.

The *OutputCtl* algorithm disables reading from the downlink FIFO buffer until a header is available. Upon receipt of the *hdr_v* signal, the *OutputCtl* algorithm transitions from a waiting state to a state that increments a counter variable. In this counting state, reading from the FIFO buffer is still disabled. This ensures that all potential delays between successive ROI do not impact the final downlink.

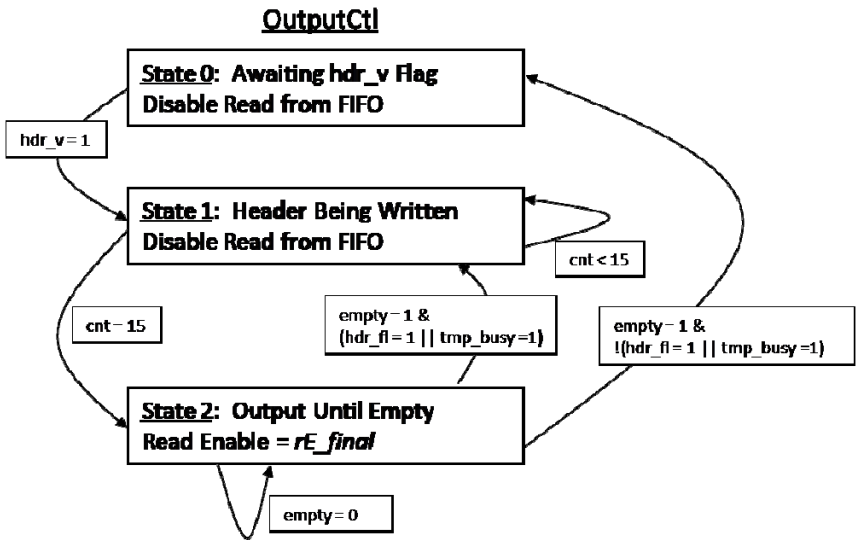

Figure 31. State Transition Diagram for *OutputCtl* Algorithm.

After counting for 15 clock cycles, the algorithm transitions to its output state. In this state, the external *rE_final* signal is forwarded to the FIFO *re* signal. The algorithm remains in this state until the downlink FIFO buffer is empty. If the *hdr_v* or *tmp_busy* flags are asserted, the algorithm immediately transitions to its counting state. Otherwise, the algorithm transitions back to the waiting state.

3. Timing Analysis

The new output format actually increases the amount of time required to generate the first header output (t_{hdr}). This is because the modified *out_hdr* algorithm needs to read and process each passed bin before generating the header. The difference is inconsequential because this time is then removed from the expression for t_{frame}. In the following expressions, the FFT latency is indicated by the term L_{FFT}. For a system using FFTv4.1, the expression for t_{hdr} is updated to

$$t_{hdr} = L_{FFT} + \overbrace{N(M-1)+2}^{\text{Time Window}} + \overbrace{\frac{N}{2}+1+\sum_{x=0}^{\#\text{ROI}-1} F(x)+1}^{\text{Frequency Window}} + \overbrace{2+\#\text{Passed Bins}}^{\text{Header Generation}}. \quad \text{(V.9)}$$

For a system using FFTv1.0, the expression for t_{hdr} is updated to

$$t_{hdr} = L_{FFT} + \overbrace{4N(M-1)+2}^{\text{Time Window}} + \overbrace{\frac{N}{2}+1+\sum_{x=0}^{\#\text{ROI}-1} F(x)+1}^{\text{Frequency Window}} + \overbrace{2+\#\text{Passed Bins}}^{\text{Header Generation}}. \quad \text{(V.10)}$$

The expression for t_{frame} is updated to

$$t_{frame} = t_{hdr} + 2 + \overbrace{M\sum F(\text{Passed Bins})}^{\text{Transmit Passed Bins}} + \overbrace{\#\text{Passed Bins}-1}^{\text{Delay Between Successive Bins}}. \quad \text{(V.11)}$$

As discussed in Chapter III, these expressions indicate the time required to write information to the downlink FIFO buffer. Since reading from the FIFO buffer is directly dependent on an external signal, no further timing analysis is required.

D. OPTIMAL MEMORY CONFIGURATIONS

This chapter examines how the circuit could be changed through understanding of the generalizations made in Chapter III. This section shows the effectiveness of these changes by demonstrating the maximum resource utilization in example configurations for the Virtex™-IIP and the Virtex™-I FPGA devices.

1. Virtex™-IIP Implementation

The SDR design was configured using the FFTv4.1 IP with the memory configurations shown in Table 6. For this example, the circuit was configured with $N = 1024$ and $M = 3$. The table shows the expected memory utilization for the SDR in this configuration.

A larger size downlink FIFO was desired for this configuration, preferably the maximum permissible FIFO depth of 64k. However, the maximum bit width permissible for this depth is 32 bits [16]. The FIFO depth was set to 1024, which permitted compilation.

The total memory required should be approximately 340 kB, which constitutes 21.5 % of the resources available on the Virtex™-IIP FPGA. This would indicate that there is significant room to increase *N* and *M* if additional functionality is desired on this device.

Purpose	Depth	Bit Width	Memory Expectation
FFTv4.1	UNK	UNK	288 kB
Time Windowing FIFO	512	76	4864 Bytes
Freq Windowing RAM	512	52	3328 Bytes
Temp Storage RAM	4096	35	17.5 kB
Temp Storage RAM	4096	35	17.5 kB
Downlink FIFO	1024	70	8960 Bytes
		Total Memory Expectation	340 kB

Table 6. Example Memory Expectation Using FFTv4.1 with $N = 1024$ and $M = 3$.

The circuit was generated to the HDL Net-list level using System Generator. The circuit was synthesized using the Xilinx ISE Project Navigator. The resulting resource estimation far exceeded the predicted amount, as shown in Table 7. The circuit uses 56 18 kB blocks of BRAM, corresponding to 1008 kB total memory usage. In this case, the

disparity between the predicted value and the one generated through synthesis does not prevent the circuit from being compiled. This anomaly is explored in more detail with the Virtex™-I implementation.

Resource	Used	Available	Percent Used
Slices	6267	9792	68%
Flip Flops	10486	19584	53%
4 input LUTs	9234	19584	47%
Bonded IOBs	96	552	17%
BRAM	56	88	63%
MULT 18X18	56	88	63%
GCLK	1	16	6%

Table 7. Resource Estimation for SDR design [From 20].

2. Virtex™-I Implementation

As discussed in Chapter II, the FFTv1.0 IP using triple memory configuration with $N = 1024$ uses 75% of the memory resources on a Virtex™-I FPGA. Even with the memory saving measures discussed earlier in this chapter, it is not feasible to fit a functioning SDR design on a single Virtex™-I FPGA using the FFTv1.0 IP where $N = 1024$. The resources required in an example where $M \leq 4$ are shown in **Error! Reference source not found.**

In order to further constrain the memory requirement, full precision was not used for the bin energy calculation. In the initial SDR design, all arithmetic calculations added bits to the output data word to prevent overflow. This level of precision is unnecessary because the signal only needs to indicate if there is enough energy to pass the threshold. Therefore, the data signal only needs to have enough precision for the minimum threshold. If arithmetic operations result in overflow, the signal can simply be saturated at its maximum value.

Purpose	Depth	Bit Width	Memory Expectation
FFTv1.0	UNK	UNK	12 kB
Time Windowing FIFO	512	16	1 kB
Freq Windowing RAM	512	16	1 kB
Temp Storage RAM	2048	16	4 kB
Temp Storage RAM	2048	16	4 kB
Downlink FIFO	1024	32	4 kB
		Total Memory Expectation	26 kB

Table 8. Example Memory Configuration Using FFTv1.0 IP.

If a multiple-FPGA system is available as the target device, then the design could be distributed between each of the devices. As discussed in Section B, the FFTv1.0 must be compiled separately from the remainder of the circuit. This neatly divides the design into two pieces that could each fit on its own Virtex™-I FPGA. System Generator compiled the compression-only portion of the circuit, creating a Xilinx ISE project. The circuit was synthesized through the Xilinx ISE project navigator. The resulting resource estimation showed that the circuit required six kB of BRAM more than the expected value.

In order to produce a configuration that could plausibly run the circuit was further divided. Bin Analysis functions were assigned to one FPGA. Temporary Storage and Downlink Control went to another FPGA, creating a three-FPGA configuration. The new configuration is shown in Figure 32.

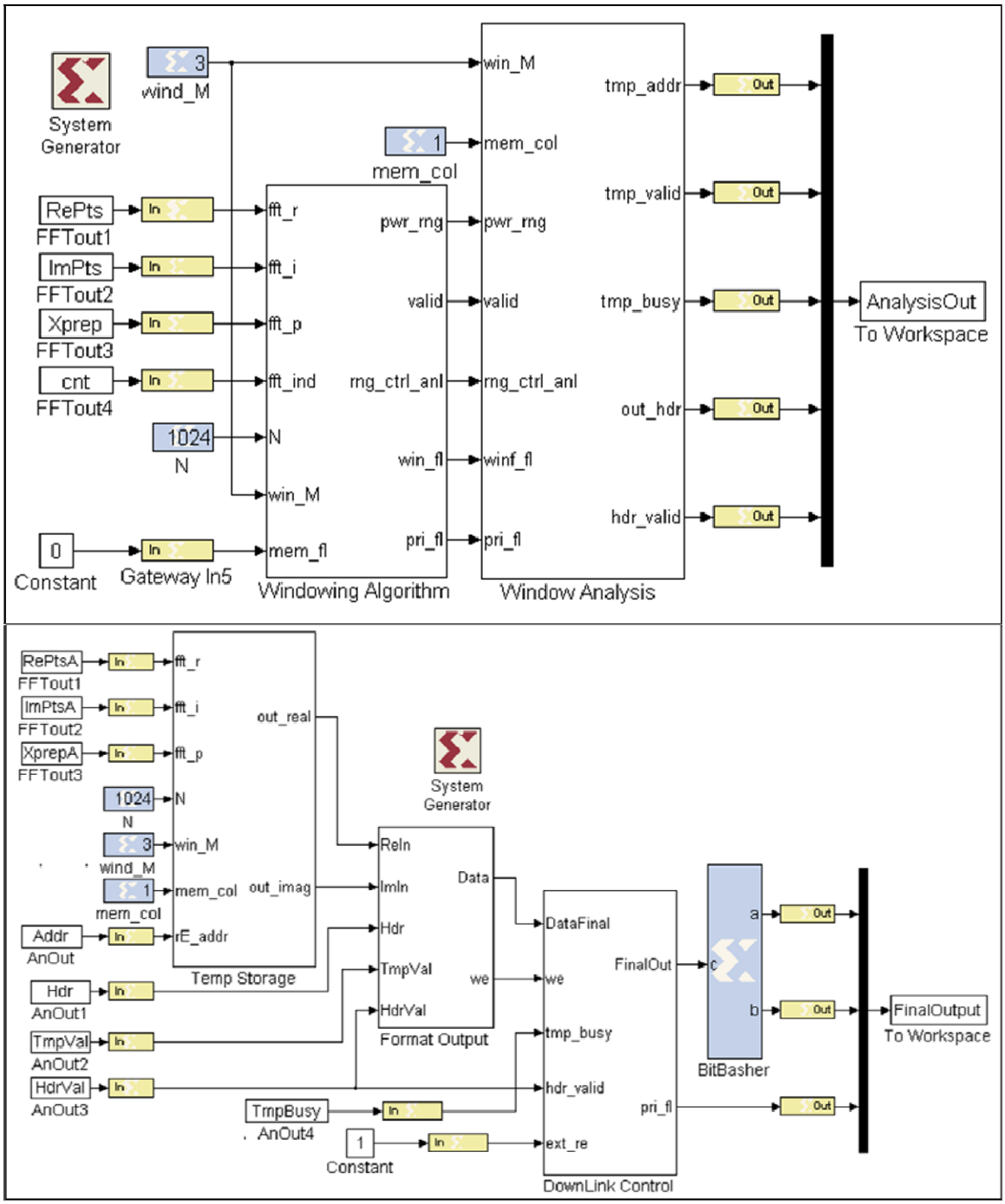

Figure 32. Partitioned Compression Algorithm for a Three-FPGA Configuration [After 3].

System Generator was used to create a Xilinx ISE project for each part of the compression algorithm. Both designs were synthesized using Xilinx ISE Project

Navigator to produce resource estimates. The bin analysis portion is shown in Table 9. The temporary storage and downlink control portion is shown in Table 9. The Temporary Storage and Downlink Control partition met expectations, with 24 BRAM blocks used. At 512 bytes for each block, this means that 12kB was used, which correlates with the information presented in Table 10. The Temporary Storage RAM and Downlink FIFO should use 12kB of BRAM.

Resource	Used	Available	Percent Used
Slices	859	12288	6%
Flip Flops	344	24576	1%
4 input LUTs	1525	24576	6%
Bonded IOBs	91	404	22%
BRAM	16	32	50%
GCLK	1	4	25%

Table 9. Resource Estimation for Bin Analysis [From 20].

Resource	Used	Available	Percent Used
Slices	109	12288	< 1%
Flip Flops	63	24576	< 1%
4 input LUTs	175	24576	< 1%
Bonded IOBs	114	404	28%
BRAM	24	32	75%
GCLK	1	4	25%

Table 10. Resource Estimation for Temporary Storage and Downlink Control [From 20].

The extra six kB memory requirement comes from the Bin Analysis portion of the circuit, as shown in Table 9. The Time Windowing FIFO and Frequency Windowing

RAM should only require one kB of memory each. Further analysis to isolate the exact cause of this anomalous memory requirement is not included within this thesis work.

E. SUMMARY

In this chapter, the information discussed in Chapter III was used to make changes to the initial SDR design. A memory and bandwidth conserving measure was implemented that takes advantage of the conjugate symmetry property of Fourier Transforms. The FFTv4.1 IP was replaced with the FFTv1.0 IP, creating a configuration that can be used on a Virtex™-I FPGA. The output format was adjusted to provide better cueing to an external communications system and make more efficient use of bandwidth. Finally, the effectiveness of these changes was demonstrated by compiling the design for both the Virtex™-IIP FPGA and the Virtex™-I FPGA. The next chapter discusses measures that increase the fault tolerance of the SDR design, making it more suitable for the space environment.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FAULT DETECTION

Ultimately, this SDR design is intended for the space environment. As discussed in [25], this means that the circuit will be vulnerable to Single Event Upsets (SEU). A SEU occurs when a high-energy particle causes a one-time bit flip, either in memory or in the output of a combinational circuit. As discussed in [6], SEUs have special implications for FPGA devices. If the SEU occurs on the output of combinational logic or in data memory then the effect is transitory. If the SEU occurs in the FPGA configuration memory, the circuit configuration will change. This may produce continuous errors, or may only produce errors for a specific input set depending on the location of the configuration fault.

This chapter explores ways of detecting faults in the circuit. Fault detection flags are communicated to the ground along with the output data. If the fault is singular, the ground user may elect to discard the data. If the fault is continuous, the user may elect to reload the FPGA configuration to remove the fault. Alternatively, the SDR controller in space may be designed to reload the configuration after a certain number of repeated errors.

A. CONSIDERATIONS

1. SDR Considerations

As discussed in [25], the vulnerability of a circuit to SEUs is dependent on its area. Large circuits are more susceptible to SEUs than small circuits. In implementing fault detection for the SDR design, the portions of the circuit requiring the largest use of combinational logic and memory were selected for fault detection algorithms. As discussed in Chapter III, the FFT IP represents the largest use of both combinational logic and memory.

As discussed in [5], Triple Modular Redundancy (TMR) presents one means of error detection and correction. Three copies of the circuit make the same calculation.

Their results are compared by a majority voter. If one of the solutions does not correlate to the other two, an error has occurred and the erroneous output is discarded. As discussed in Chapter IV, neither the Virtex™-IIP nor the Virtex™-I FPGA would have room to accommodate two additional copies of the FFT IP.

Snodgrass introduced the concept of Reduced Precision Redundancy (RPR) in [5]. If a less-precise numerical solution is acceptable, then two low-precision copies of the circuit could be used to generate an upper and lower bound. If the precise solution is outside the bounds, an error has occurred. In that case, the average between the upper and lower bound is used instead of the precise output.

While RPR looks like a feasible means of producing a fault correction by trading precision for area, it would not produce significant memory savings for the FFT IP used in this design. The FFT IP prevents any modification to the internal circuitry beyond the configuration options presented to the user. This prevents the use of either RPR or TMR for low-level calculations. As discussed in [22], the FFTv1.0 phase factor bit width is fixed at 16 bits. Intermediate values are stored at this level of precision, independent of the input data's precision. The number of memory blocks required for the FFTv4.1 IP is specifically listed in [15], grouped by target device and configuration options. As with the FFTv1.0 IP, the amount of memory required is independent of the input precision.

For this SDR design, neither TMR nor RPR seem to be feasible options for detecting errors that occur within the FFT IP. In order to work with the SDR design, fault detection algorithms must use only the remaining chip space shown in Chapter IV. A simple computation relating the FFT input to the FFT output is desired.

2. Parseval's Theorem

As discussed in [9], the Discrete Fourier Transform has the inner product property, also known as Parseval's Theorem where

$$\sum_{n=0}^{N-1} x^*[n]y[n] = \frac{1}{N}\sum_{k=0}^{N-1} X^*[k]Y[k]. \qquad \text{(VI.1)}$$

This equation relates the input to the FFT directly to the output of the FFT in a way that is much less computationally intensive than the FFT algorithm. Because this expression does not produce a duplicate FFT output it cannot be used for error correction, but it can be used for error detection. Both sides of the equation require N complex multiplication operations and N complex addition operations.

Equation (VI.1) must be adjusted so that it can be applied in this design. The input to the FFT is a real signal. The term $y[n]$ is replaced with $x[n]$. The left side of the expression is adjusted to

$$\begin{aligned}\sum_{n=0}^{N-1} x^*[n]y[n] &= \sum_{n=0}^{N-1} x^*[n]x[n] \\ &= \sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\} + \mathrm{Im}^2\{x[n]\} \\ &= \sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\}\end{aligned} . \qquad \text{(VI.2)}$$

As discussed in Chapter II, the output of the FFT is already scaled by a factor of $1/N$. The right side of the equation is adjusted to use the real and imaginary portions of the signal. Then an additional scaling factor of $1/N$ is included on each side. This produces an expression that can be implemented using the FFT input and output, where

$$\begin{aligned}\sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\} &= \frac{1}{N}\sum_{k=0}^{N-1} X^*[k]X[k] \\ \sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\} &= \frac{1}{N}\sum_{k=0}^{N-1} \mathrm{Re}^2\{X[k]\} + \mathrm{Im}^2\{X[k]\} \\ \frac{1}{N}\sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\} &= \sum_{k=0}^{N-1}\left(\frac{\mathrm{Re}\{X[k]\}}{N}\right)^2 + \left(\frac{\mathrm{Im}\{X[k]\}}{N}\right)^2\end{aligned} . \qquad \text{(VI.3)}$$

Although Equation (VI.3) can be easily implemented in hardware, it presents a timing problem when used with this design. The left side of the expression can be calculated before the FFT circuit has completed its computation. As discussed in Chapter IV, the compression portion of the design now uses only the first $N/2$ points of the FFT

output. Depending on the size of the user-defined ROI, the circuit might not be able to complete computation of the right side of Equation (VI.3) before the header is ready for downlink.

The conjugate symmetry property shown in Equation (V.1) and Equation (V.2) can also be used to reduce the right side of Equation (VI.3). This is shown in the expression

$$\sum_{k=0}^{N/2-1} X^*[k]X[k] = \sum_{k=N/2}^{N-1} X[k]X^*[k]. \qquad \text{(VI.4)}$$

This means that Equation (VI.3) can be adjusted to

$$\frac{1}{N}\sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\} = 2\sum_{k=0}^{N/2-1}\left(\frac{\mathrm{Re}\{X[k]\}}{N}\right)^2 + \left(\frac{\mathrm{Im}\{X[k]\}}{N}\right)^2. \qquad \text{(VI.5)}$$

By using Equation (VI.5), the number of arithmetic operations required to calculate the right side of the expression is reduced by half. This ensures that both sides of the expression can be calculated and compared in time to include error detection information in the header for each time window data frame.

3. Parity Checking

As discussed in [23], a parity bit can be used to detect corruption of data. For even parity a bit is appended to the data word to ensure that the number of ones in the word is even. If the number of ones in the data word is even, the parity bit will be zero. If the number of ones in the data word is odd, the parity bit will be one. Wakerly shows that the Exclusive Or (XOR) function can be used to determine the appropriate value of the parity bit [23]. If an odd number of errors occur in the data word, the parity bit will detect that an error is present. If the number of errors in the word is even, they will not be detected by a single parity bit.

B. MODIFICATIONS TO DESIGN

This section discusses how the design was modified to add fault tolerance. The baseline for changes is the example Virtex™-IIP configuration shown in Chapter IV.D.1, where $M = 3$ and $N = 1024$. This model incorporates the changes discussed in Chapter IV. The Virtex™-IIP configuration was selected because using only one chip was desirable to facilitate signal routing while adding new features to the design. Parseval's Theorem was used to detect errors in the FFT computation. Parity bits are used to detect errors in other memory blocks within the circuit.

1. Error Checking Using Parseval's Theorem

An error in the FFT computation is detected by calculating the left and right sides of Equation (VI.5) independently. If the results differ by a predetermined threshold, an error flag for the FFT time window is forwarded to the header generation algorithm *out_hdr*. The header was adjusted to forward error flags for the time windows.

a. Implementation

The FFT subsystem was adjusted to calculate the left side of Equation (VI.5), as shown in Figure 33. A multiplier IP block is used to calculate $\mathrm{Re}^2\{x[n]\}$. The subsystem labeled *Accum* developed in [3] uses a pair of accumulators to continuously sum the streaming input. This subsystem was used because of its demonstrated reliability. It is possible that the accumulator subsystem could be replaced with a simpler configuration using only one accumulator. The accumulator would need to be configured to reinitialize with the current input on reset, as discussed in [24]. This would permit the system to calculate the sum of continuous inputs using only one accumulator.

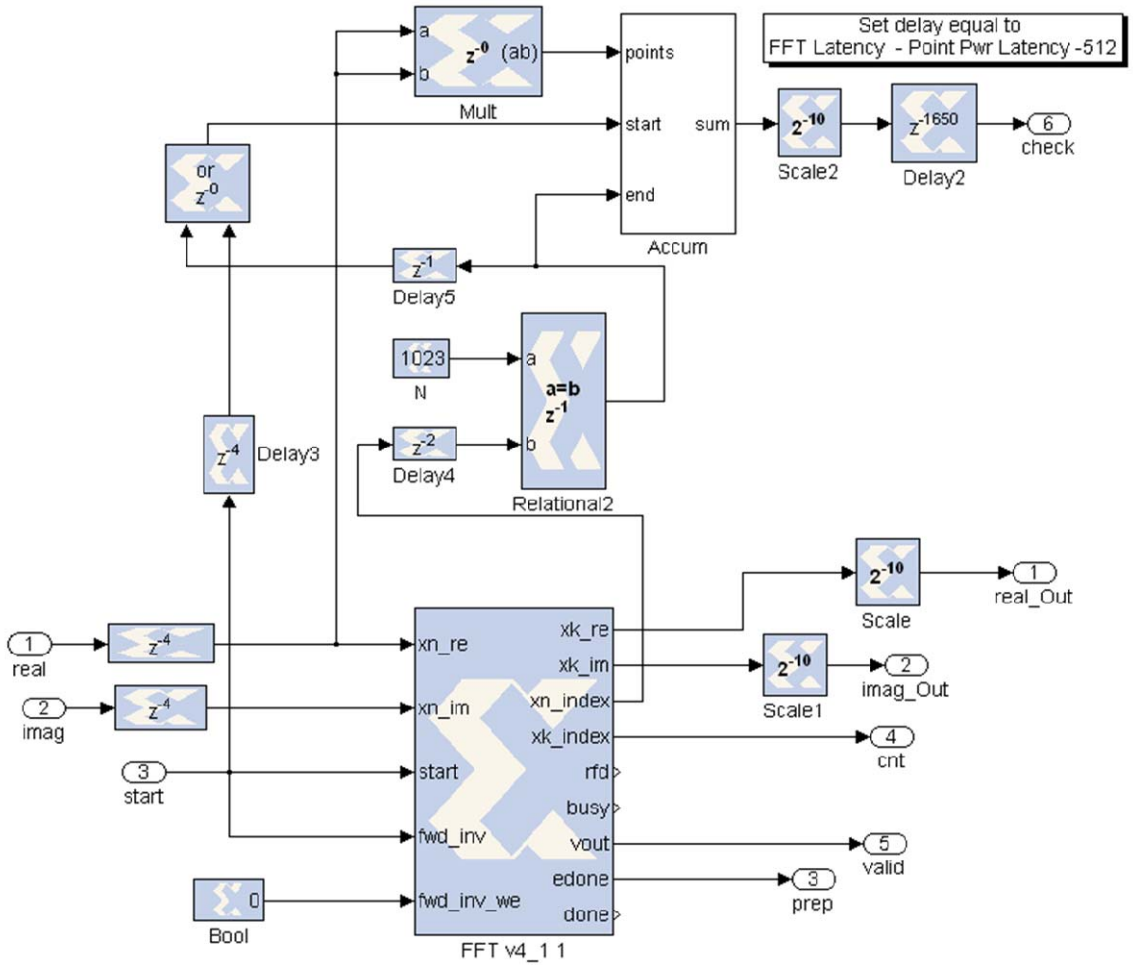

Figure 33. FFT Subsystem Modified for Error Detection [After 3].

The system relies on timing signals based on the FFT v4.1 IP to indicate the beginning and end of each FFT period. The *xn_index* signal corresponds to the index of the input points. This value is compared to 1023 to detect the end of the current input period. This value is delayed by one clock to indicate the start of the next period. The circuit would need to be adjusted for the FFTv1.0 IP by using a counter to simulate the *xn_index* signal.

The FFT subsystem scales the accumulator output by $1/N$ then delays the signal to align it with the FFT output. The value $\sum_{n=0}^{N-1} \mathrm{Re}^2\{x[n]\}$ leaves the FFT subsystem as the *check* signal. This signal is routed to the Time Windowing subsystem. As

discussed in Chapter III, the SDR compression algorithm determines the energy in each FFT point by calculating $\left(\mathrm{Re}\{X[k]\}/N\right)^2+\left(\mathrm{Im}\{X[k]\}/N\right)^2$. This value is routed into a new FFT Error Detection Subsystem, as shown in Figure 34. The *e_done* and *p_cnt* signals are also routed to the FFT Error Detection Subsystem for timing purposes. As discussed in Chapter III, these signals indicate the start of the FFT output period and the current FFT output index, respectively.

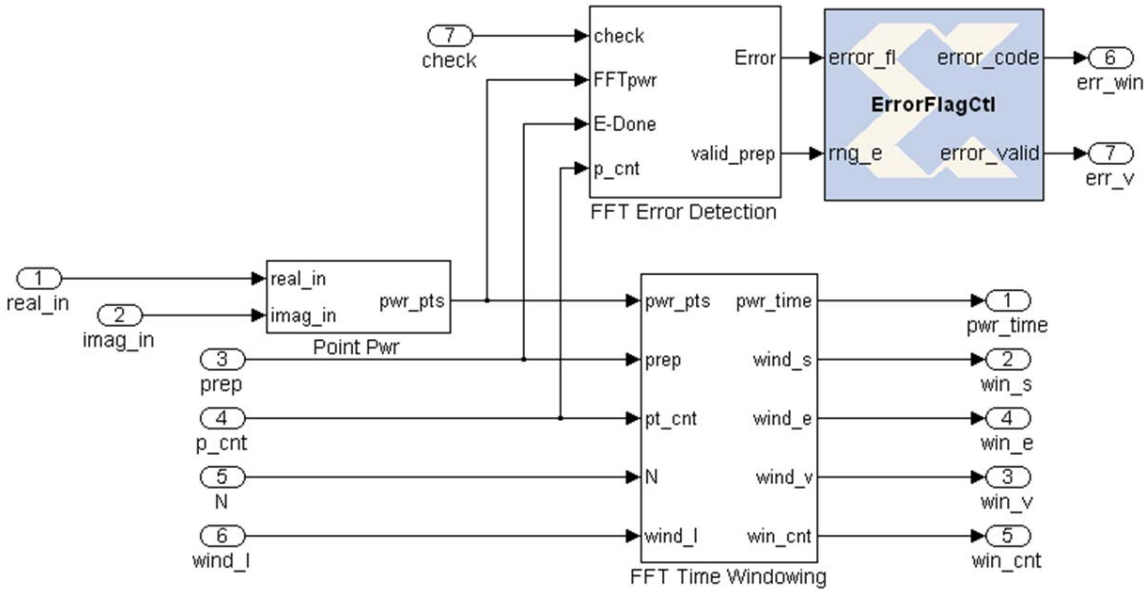

Figure 34. Modification for FFT Error Detection [After 3].

Within the FFT Error Detection Subsystem, another copy of the Accumulator subsystem is used to calculate the sum of the *pwr_pts* signal, as shown in Figure 35. The sum is scaled by a factor of two completing all calculations required for Equation (VI.5). The difference between the left and right sides of the equation is calculated using a subtraction IP block. The difference is compared with a positive and negative threshold. If both comparisons yield a true result, then no error exists. In any other case, the *Error* flag will be set. The *e_done* signal is used to indicate the start of an FFT output sequence. The *p_cnt* signal is used to determine when enough points have

been collected to calculate the sum on the right side of Equation (VI.5). When this point in time is reached, the *valid_prep* flag is set to indicate that the *Error* flag will be valid on the next clock cycle.

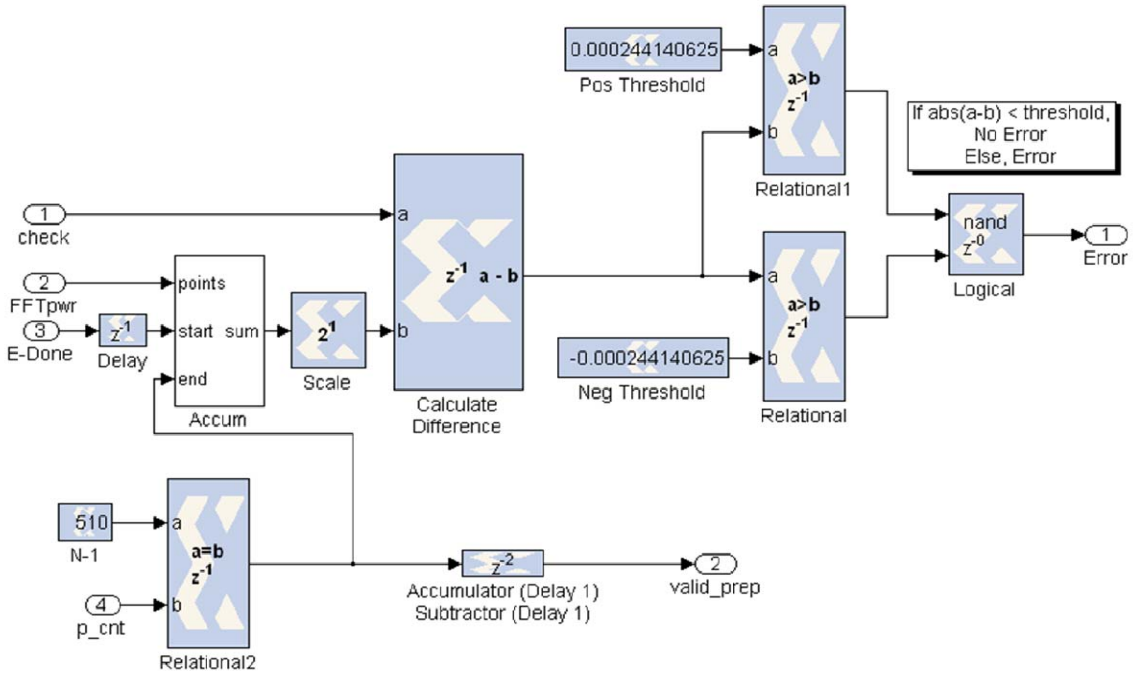

Figure 35. FFT Error Detection Subsystem.

The *Error* flag and *valid_prep* flag are passed to the *ErrorFlagCtl* algorithm. This algorithm uses the *Error* flags from each FFT period within the user-defined time window to generate an error code, indicated by the *err_win* signal in Figure 34. The length of the error code is equal to *M*, the number of FFT periods in each time window. Each bit is a flag that indicates whether or not an error was detected within that FFT period. The state transition diagram for the *ErrorFlagCtl* algorithm is shown in Figure 36.

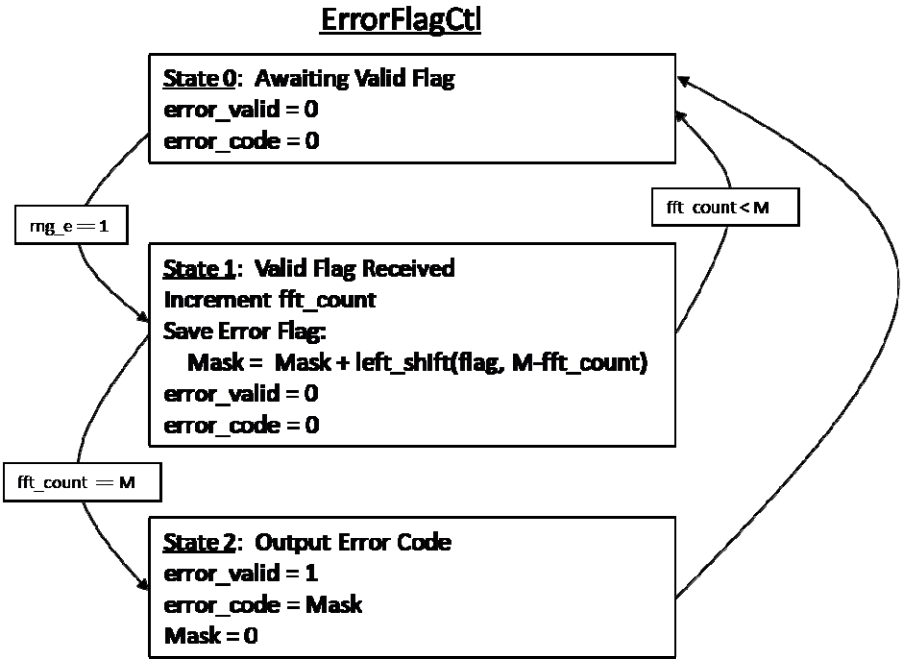

Figure 36. State Transition Diagram for the *ErrorFlagCtl* Algorithm.

The error code is forwarded to the Header Generation Subsystem to be included in the header for each time window frame. An additional FIFO buffer was added to the subsystem to store the error code until the completion of the bin energy analysis. The header format was adjusted to include the error code, as shown in Figure 37.

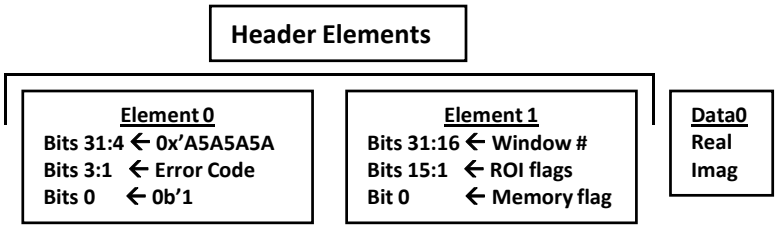

Figure 37. Header Format with Error Code.

b. Testing

A small set of tests was conducted to ensure the circuit functions as desired, and to determine an appropriate value for the threshold. The initial error

threshold was set to 2^{-10}. The FFT error checking measures were first tested with no additional adjustments to the circuit. As expected, an error code of "000" was generated indicating that no errors were detected.

Next, a multiplexer was inserted into the path of the FFT output corresponding to $\mathrm{Re}\{X[k]\}/N$, as shown in Figure 38. The error injection circuit compares the output of a free-running counter with the constant 3000 to determine if the correct value will be forwarded or a constant error. Looking at the first time window, this means that the first FFT period is error-free, while the second and third will contain errors. The circuit failed to detect the error using the initial threshold. When the threshold was lowered to 2^{-12} the expected error code of "011" was generated. The code was properly forwarded to the header generation algorithm, and was included in the data frame sent to the downlink.

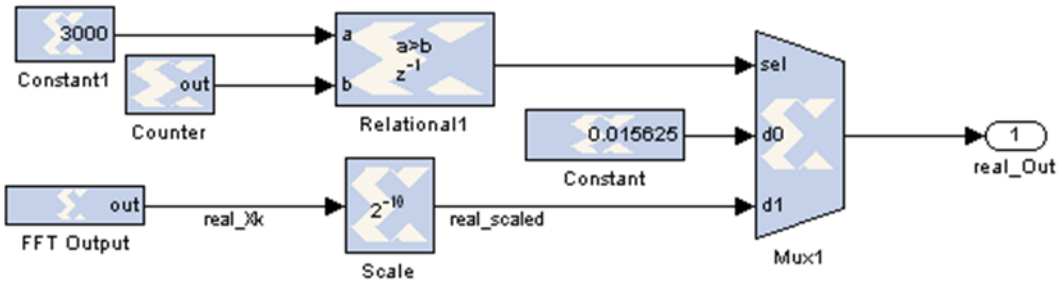

Figure 38. Error Injection Circuit for FFT Output.

2. Memory Error Detection

As discussed in Section A, parity can be used to detect errors in memory. The most likely place for a memory error in this circuit is in the Temporary Storage subsystem. This subsystem has the largest memory requirement in the circuit outside the FFT IP. It is also has the longest temporal storage requirement. As discussed in Chapter III and Chapter IV, information could be held for up to *MN* clock cycles. For these reasons, the Temporary Storage Subsystem was selected to as the best location to implement a memory error detection algorithm.

a. Implementation

As discussed in Chapter III and Chapter IV, the signals stored in the Temporary Memory Subsystem are 35-bit numbers when using the Virtex™-IIP configuration. In order to generate parity bits for each incoming number, the circuit must evaluate the expression

$$P = \text{XOR}\left\{\text{Bit}[34], \text{Bit}[33], ...\text{Bit}[1], \text{Bit}[0]\right\} \tag{VI.6}$$

To implement Equation (VI.6), a tree of Bit Basher blocks is placed in series with a tree of XOR gates, as shown in Figure 39.

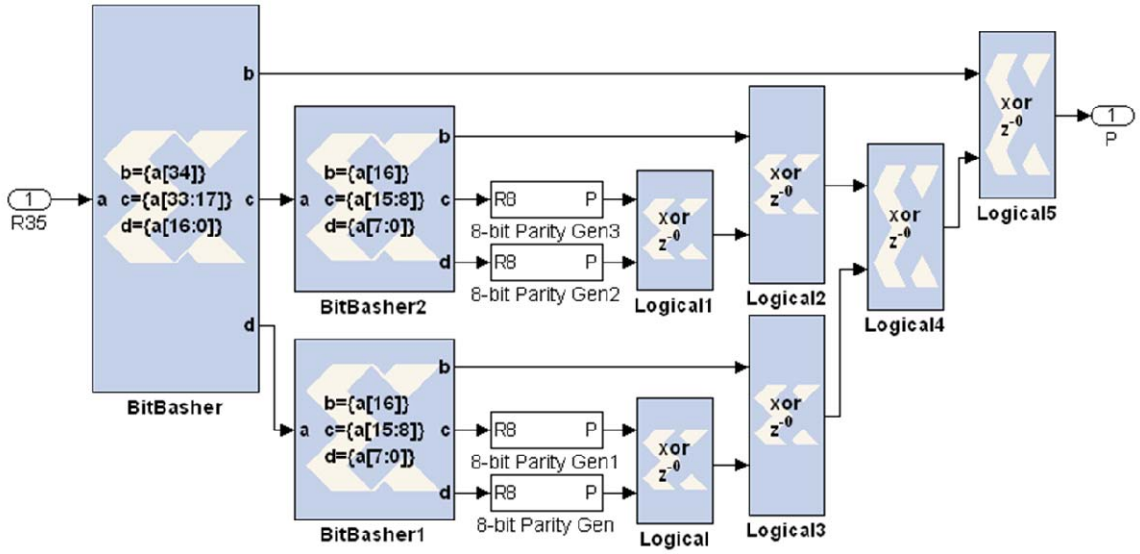

Figure 39. 35-bit Parity Generator.

As discussed in [24], the Bit Basher blocks require no hardware overhead. They are used here as a means of re-interpreting a single multiple-bit signal as multiple signals with smaller bit widths. Each input signal can only be divided four ways, so a tree of Bit Basher blocks is required to re-interpret a single 35-bit signal as 35 one-bit signals. This method of error detection will also work with the Virtex™-I configuration, provided a parity generator for a 16-bit data word is used.

The 35-bit parity generator was added to the Temporary Memory Subsystem to generate a one-bit parity code from the FFT point entering memory. This is

shown for the real portion of the signal in Figure 40. The parity bit is stored in a separate Dual-Port RAM using the same addressing signals as the Dual-Port RAM for the data word. On the output from the Dual-Port RAM, a new parity bit is generated and compared with the one stored in memory using an XOR gate. If the bits do not match, then an error has occurred. This design is duplicated for the imaginary portion of the signal.

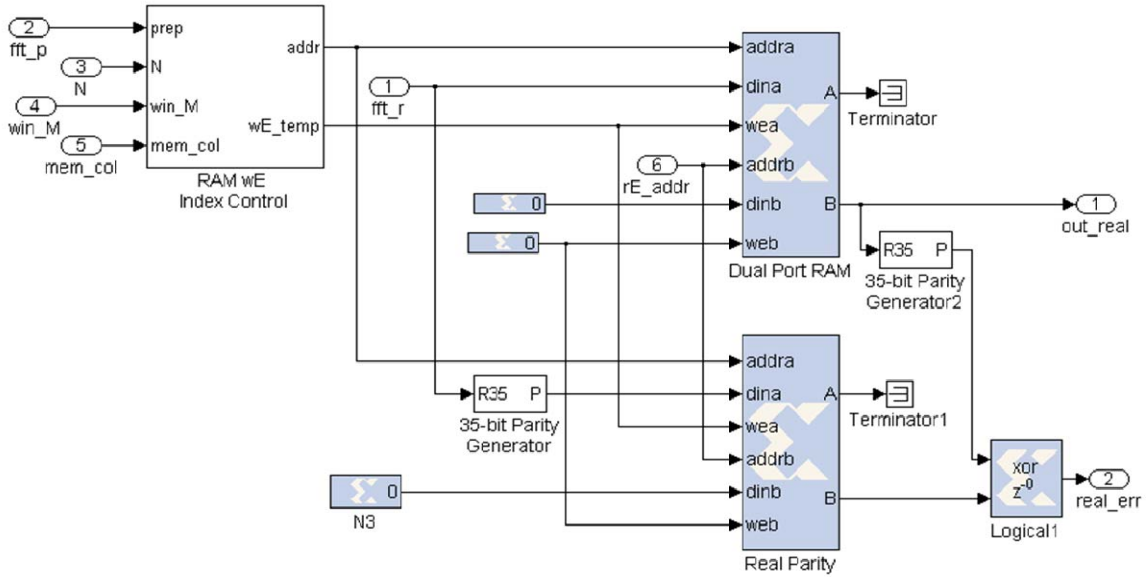

Figure 40. Modification for Memory Error Detection (After: [3]).

The error flags are forwarded to the Format Output Subsystem. A finite state machine was created to accumulate error flags and generate an error code that is appended to the end of the downlink data frame. The state transition diagram for the *ParityFlagCtl* algorithm is shown in Figure 41. In State Zero, the algorithm waits for output from the Temporary Memory Subsystem. An initial error code of three was used for troubleshooting purposes.

When the *tmp_busy* flag is asserted, valid output from the Temporary Memory subsystem will be available on the next clock cycle. The error code is shifted left by two bits to make room to record error flags and the algorithm transitions to State Two. Errors in the real portion of the signal are recorded in the left bit. Errors in the imaginary portion of the signal are recorded in the right bit. The error code saves one

error from each ROI selected for transmission. If an error is recorded, then a flag is set to prevent additional errors from interfering with the values stored in the error code. When the algorithm detects that a new ROI is about to be read from Temporary Memory, it shifts the error code left by two bits to make room for the next set of flags.

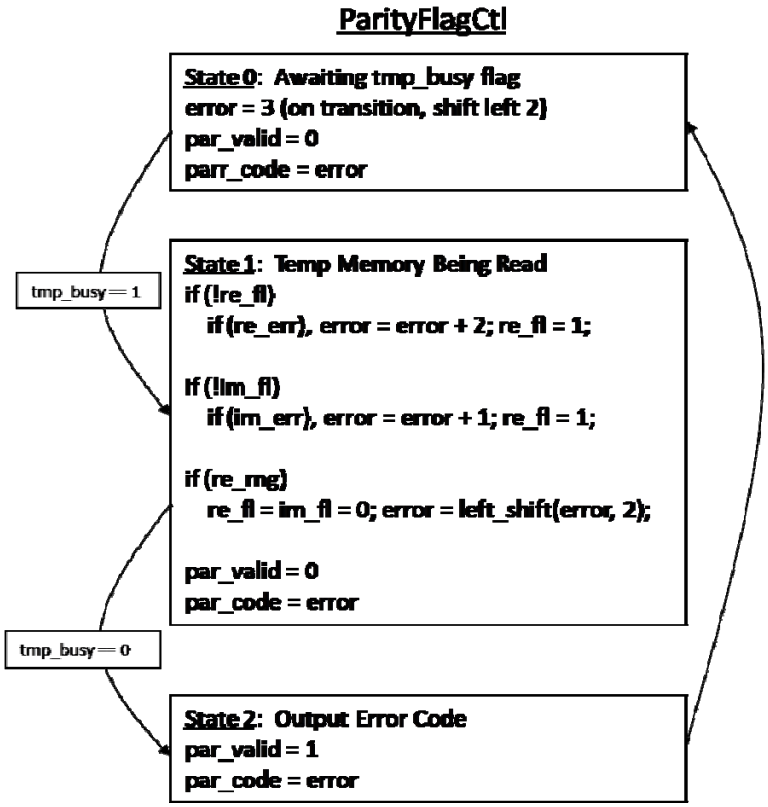

Figure 41. State Transition Diagram for the *ParityFlagCtl* Algorithm.

When the *tmp_busy* transitions to zero, all the values for the current data frame have been read from Temporary Memory. The *ParityFlagCtl* Algorithm transitions to State Two and sets the *par_valid* flag, indicating that the error code is valid. On the next clock cycle, the algorithm transitions back to State Zero to await the next set of values from Temporary Memory.

The *ParityFlagCtl* Algorithm was inserted in the Format Output Subsystem, as shown in Figure 42. The logic gate used to control write enable for the

Downlink FIFO was adjusted so the *par_valid* signal would also enable writing to the FIFO. An additional multiplexer was added to the FIFO input path, allowing the *par_code* signal to be added to the data frame.

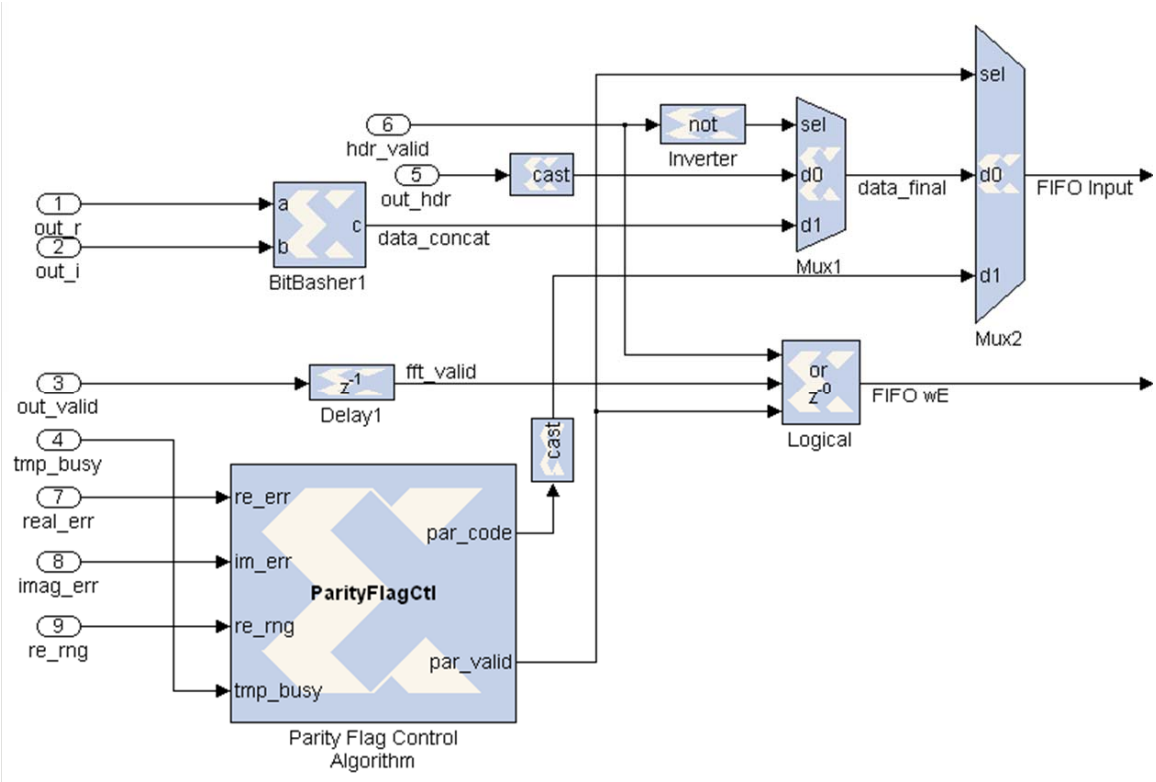

Figure 42. Modification to Communicate Memory Errors [After 3].

As discussed in Chapter IV, the maximum number of user defined ROIs is set at 15. The *par_code* signal is a 32-bit number, which accommodates the maximum number of ROIs. The format of the signal is shown in Table 11. The code must be interpreted using the number of ROIs that were selected for transmission. As discussed in Chapter IV, this information is included in the data frame header.

	15 ROI Transmitted	One ROI Transmitted
par_code[31:30]	11	00
par_code[29:28]	ROI0[real error, imag error]	00
par_code[27:26]	ROI1[real error, imag error]	00
…	…	…
par_code[3:2]	ROI13[real error, imag error]	11
par_code[1:0]	ROI14[real error, imag error]	ROI0[real error, imag error]

Table 11. Format of the *par_code* Signal.

b. Testing

The parity checking algorithms were tested using an input data set that was selected to ensure that two frequency bins would be selected for downlink. On the first test, no errors were injected. As expected, the output *par_code* signal was 48_{10}, or 110000_2, indicating that no error was detected. The initial error code of three was shifted left by two bits when each successive frequency bin was sent to the Downlink FIFO.

On the second test the output of the imaginary Dual-Port RAM was multiplexed with an error injection circuit, similar to the one shown in Figure 38. A constant error was injected when the free-running counter exceeded 4767 clock cycles. This time was selected to ensure that the second frequency bin would contain errors, but the not the first frequency bin. As expected, the output *par_code* signal was 49_{10} or 110001_2, indicating that an error was detected in the imaginary portion of the second frequency bin.

3. Resource Check

The SDR circuit with error detection capability was compiled using System Generator to generate a Xilinx ISE project. Xilinx ISE Project Navigator was used to synthesize the design. The results of the synthesis are shown in Table 12. The table also

displays the increase in percentage of device resources used in comparison with the baseline design without an error detection capability. As expected, the memory utilization only increases slightly. The error detection algorithms require a considerable increase in the number of resources used for combinational logic, but the design IS still able to fit on a Virtex™-IIP FPGA.

Logic	Used	Avail	Pct Avail	Pct Increase
Slices	9304	9792	95 %	27 %
Flip-Flops	14925	19584	76 %	23 %
4-input LUTs	14234	19584	72 %	25 %
IOBs	96	552	17 %	0 %
BRAMs	59	88	67 %	4 %
MULT 18x18	61	88	69 %	6 %
GCLK	1	16	6 %	0 %

Table 12. Virtex™-IIP Resources Required for Error Detection (From: [20]).

C. CONCLUSIONS

This chapter discussed the requirements of a circuit designed for the space environment. Various options for implementing fault tolerance were explored. The use of Parseval's Theorem to check for errors in the FFT computation was introduced, implemented, and tested. Parity-checking algorithms were added to detect faults in the Temporary Memory Subsystem. Algorithms were added to communicate errors in the output data frame. The resulting design was synthesized, ensuring that the SDR design would still fit on a Virtex™-IIP FPGA. The next chapter presents a summary of this thesis work and provides recommendations for future work.

VI. CONCLUSION

This chapter presents a summary of the objectives achieved through this review of the initial SDR design. Recommendations for future work are provided.

A. CONCLUSIONS

The methods used for Fourier Analysis were reviewed. The timing and resource requirements of the FFTv4.1 and FFTv1.0 IP circuits provided by Xilinx were examined using a configuration with $N = 1024$. The information provided in [15] and [22] was verified through simulations with DC input signals and synthesis using System Generator and the Xilinx ISE Project Navigator.

The initial SDR design presented in [3] was examined using a configuration with FFT length $N = 1024$ and the number of FFT periods per time window M set to three. Internal timing considerations were clarified using state transition diagrams and timing charts to illustrate the behavior of the circuit's control algorithms. General expressions were created regarding the circuit's timing and resource requirements for any selection of N and M. These expressions can be used as design equations to estimate appropriate values for N and M given a fixed amount of available memory on a target FPGA device.

Changes were made to increase downlink efficiency, decrease latency, and decrease memory utilization by taking advantage of the conjugate symmetry inherent in the FFT algorithm. The Format Output Subsystem was adjusted to improve signal flow to an external communications system. The downlink data frame format was adjusted to increase efficiency. One possible circuit configuration was presented that fits on a Virtex™-IIP FPGA. An alternate configuration was provided with the circuit functions distributed over three interconnected Virtex™-I FPGAs.

The circuit was made more suitable for the space environment through the addition of a fault detection capability. Options for fault detection and correction were examined. A fault detection method using Parseval's Theorem was designed and its functionality was verified. Parity checking algorithms were added to detect faults in

some of the memory banks used in the design. The format of the downlink data frame was adjusted to communicate fault status to terrestrial systems.

The changes discussed in Chapter V represent the final modifications to the design for this body of work. The final configuration was saved in the Simulink® model *SDR1024Mod8C*. Appendix A lists all associated files required for compilation, as well as the revision history of the design. Wright provided a list of items that would need to be changed to ensure the design functions under different configuration options [3]. This list is updated in Appendix A.

B. RECOMMENDATIONS

This thesis work focused on the practical implementation of the algorithm documented in [3]. To supplement the recommendations listed in [3], additional work could be done in the following areas to improve the circuit's capability and verify its reliability.

1. Bin Overlap

The current algorithms were written with the assumption that user-defined frequency bins could not overlap. Although the circuit would have no difficulty processing overlapping bins, this would lead to inefficiency in the downlink since FFT points in the overlap region would be sent twice. In order to correct this inefficiency, the bin range input to the *re_tmp* algorithm will need to be adjusted. The circuit would need to detect if bin overlap exists and determine if both overlapping bins pass the bin threshold analysis. If one or both of the overlapping bins fails the threshold analysis, the circuit functions normally.

2. Pipelining

As discussed in Chapter III, this circuit does not take advantage of the pipelining features available in the multipliers and adders. The compression portion of the circuit uses two adders, two multipliers, and two accumulators. Pipelining these arithmetic operations would lower the clock period, increasing the sample rate along with the

sensitivity of the circuit. This change would require some adjustments to the timing algorithms. The *pwr_time* algorithm and *accum_ctl* algorithm are the most likely to be affected by pipelining. The delays inserted for timing purposes would also need to be adjusted to accommodate pipelining.

3. Comprehensive Test Set

The modifications made in this thesis work were tested using a small range of possible inputs and user-defined configurations. The work documented in [3] used a larger range of tests. Even these tests did not come close to testing the algorithm under its most stressful conditions. The circuit needs to be tested with the user-defined ROI maximized over a period of time that would confirm its ability to gracefully overwrite obsolete memory. Additionally, it needs to be tested with input signals that are not tuned to the sampling rate of the FFT. The fault detection algorithms should be tested with a wider range of faults and different fault thresholds to determine an optimal configuration. Finally, the fault detection algorithms could be tested in a radiation environment to verify that they perform as designed.

4. Improve User Interface

The algorithm in its current form is vulnerable to user misuse through poor configuration choices. The algorithm could be adjusted to detect and prevent user-entered configurations that would cause the algorithm to crash or produce anomalous output through unintended memory overwrites. Since the procedure to set up the configuration is complex, a user guide should be developed. The compilation instructions included in Appendix A could be used as a baseline. A graphical user interface could also be developed using MATLAB® to facilitate the setup process. Finally, some decompression algorithms for the design make use of intermediate signals sent to the MATLAB® workspace for troubleshooting. Since these signals would not be available in the actual implementation, a user-friendly decompression algorithm using only the final output of the SDR circuit should be developed.

5. Explore Other Methods to Compute the FFT

This thesis explored using both the FFTv4.1 IP and the FFTv1.0 IP as the means to compute the FFT for the SDR circuit. While the FFTv4.1 IP is compatible with the Virtex™-IIP FPGA, neither of these IP circuits is compatible with the Virtex™-II FPGA. If the Virtex™-II FPGA is desired as the target device, the FFTv3.1 and FFTv3.2 IP circuits could be examined as a feasible means to compute the FFT for the SDR circuit [15], [22], [26], [27].

This thesis focused on using existing IP to compute the FFT. As discussed in Chapter V, this prevented the use of internal fault detection and correction methods such as Triple Modular Redundancy and Reduced Precision Redundancy. If this level of fault tolerance is desired for this circuit, an FFT would have to be developed in the System Generator environment using fault-tolerant algorithms within the Cooley-Tukey algorithm. An RPR version of this algorithm that could be used as a basis for an FFT circuit for the SDR design is demonstrated in [6].

APPENDIX A. IMPLEMENTATION DETAILS

This appendix provides lists of the files required to run the simulations discussed in this research. A set of instructions for simulation and synthesis of the design is provided. Additionally, the list of modules affected by changes to *N* and *M* is updated.

A. REQUIRED FILES

A new Simulink® model was saved with each major adjustment to the design. All of these files and folders used for this design are available on DVD. The NPS CRL Lab Manager can be contacted for a copy of the DVD. A list of important subdirectories is shown in Table 13. A list of m-files required to configure the MATLAB® environment for the simulation is provided in Table 14. A list of the Simulink® models representing different stages in development is provided in Table 15. A list of m-files required to implement algorithms within the m-code blocks of the initial SDR design is provided in Table 16. Files that replace or augment the ones used in the initial SDR design for follow-on models are listed in Table 17.

Sub Directory	Description
FFT_Testing	Contains all files used to test the FFTv4.1 and FFTv1.0 IP blocks as discussed in Chapter II.
DurkeInit	Contains all model and m-code files developed for the thesis work documented in [3].
DurkeInit/Mods	Contains all model and m-code files modified for the work documented in this thesis.
FFTdev	Contains models and m-code files for an initial attempt to develop an FFT algorithm in the System Generator environment using IP blocks for elementary math operations.

Table 13. Important Sub Directories Available on DVD.

File Name (.m)	Description
input_sig_gen	Creates an input signal for SDR testing [3].
input_sig_gen2	Doubles the number of frequencies in the input signal.
ROI_ctrl	Creates ROI input to SDR design [3].
Test_control_testing	File used to run tests discussed in [3].
Test_control_testing_Rev2	Modifies the input data set to focus on the first time window.
Test_control_testing_Rev3	Increases the size of user-defined ROI.
Test_control_testing_Rev4	Adjusts user-defined ROI and input signals to test $N/2$ configuration.
Test_control_testing_Rev5	Reformats model I/O signals for a 2-chip implementation.
Test_control_testing_Rev6	Reformats model I/O signals for a 3-chip implementation.

Table 14. M-Code files External to the SDR Design.

Model Name (.mdl)	**Description**
FFTv1Test3	Test the performance of the FFTv1.0 IP block.
FFTv4Test	Test the performance of the FFTv4.1 IP block.
SDR_1024_point_1	Initial SDR design, described in [3]
SDR_1024MOD2	Circuit modified for $N / 2$ compression
SDR_1024MOD3A	Compression algorithm only. FFT computation removed.
Mod3_Chip1of3	FFTv1.0 computation only. Used in conjunction with either SDR_1024MOD3A or SDR_1024Mod7Chip2B and SDR_1024Mod7Chip3B.
SDR_1024MOD4	Adds adjustments to downlink control.
SDR_1024MOD5	Reduces memory requirement of Time Windowing and Freq Windowing subsystems to minimum.
SDR_1024MOD6	Optimal configuration for Virtex™-IIP. All troubleshooting signals removed to track signal formats.
SDR_1024MOD7Chip2B	Window Analysis subsystems only. Chip 2 of a 3-chip Virtex™-I configuration.
SDR_1024Mod7Chip3B	Temporary Storage, Format Output, and Downlink Control Subsystems. Chip 3 of a 3-chip Virtex™-I configuration.
SDR_1024Mod8C	Added FFT error checking algorithms. This model includes error injection algorithms for testing.

Table 15. Simulink® Model Files Used in this Thesis Work.

Algorithm	**File Name (.m)**	**Description**
accum_ctrl	*accum_ctrl_3_1*	Manages signals to control two accumulators.
hdr_data_mgt	*hdr_data_mgt*	Manages signals passed to the *out_hdr* algorithm.
mem_pri	*mem_pri*	Sets a flag to use more a smaller set of ROIs when in a restricted memory condition.
out_hdr	*out_hdr*	Produces a header for the output data frame.
pwr_time	*pwr_time_1*	Manages Time Windowing subsystem signals. Original Design, assumes continuous FFT output.
re_freq_win	*re_freq_win_1*	Manages signals and addressing when reading user-defined ROIs from a dual-port RAM.
re_tmp	*re_tmp_1*	Manages signals and addressing when reading values out of Temporary Memory.
we_temp_fft	*we_temp_fft_1*	Manages signals and addressing when writing FFT output to Temporary Memory.
we_time_win	*we_time_win_1*	Manages signals and addressing when writing Time Window subsystem output to a dual-port RAM.
wind_anal	*wind_anal_2*	Manages the signals associated with evaluating the number of ROIs that pass threshold analysis.

Table 16. M-Code files Used in the Initial SDR Design, as Discussed in Chapter III.

Algorithm	**File Name (.m)**	**Description of Change**
ErrorFlagCtl	*ErrorFlagCtl*	Interprets errors detected in the FFT algorithm and generates an error code as discussed in Chapter V.
out_hdr	*out_hdrMod1*	Produces an efficient, fixed-length header.
out_hdr	*out_hdrMod2*	Includes the FFT error code in the header.
OutputCtl	*OutputCtlMod0*	Controls reading from the downlink FIFO buffer as discussed in Chapter IV.
ParityFlagCtl	*ParityFlagCtl*	Saves errors detected in memory through parity checking algorithms and produces a parity error code for downlink.
pwr_time	*pwr_time_MOD2*	Adjusts algorithm to interpret only $N/2$ points.
re_freq_win	*re_freq_win_Mod1*	Adjusts algorithm to interpret only $N/2$ points.
re_tmp	*re_tmp_Mod1*	Adjusts algorithm to read only $N/2$ points per FFT period.
we_temp_fft	*we_temp_fft_Mod1*	Adjusts algorithm to write only $N/2$ points per FFT period.
we_time_win	*we_time_win_Mod1*	Adjusts algorithm to interpret only $N/2$ points.
wind_anal	*wind_anal_Mod1*	Added comments to m-code for clarification.

Table 17. M-Code Files Added or Adjusted for Changes to the SDR Design.

All files must be opened on a system configured for use with the appropriate MATLAB®, System Generator, Xilinx ISE, and ModelSim® software, as listed in Chapter II. Opening the design without all software correctly configured results in unrecoverable corruption of the model file. Working from a set of backup files is

recommended until the functionality of all design tools is verified. All required files must be located within the same directory to be recognized when the simulation is run [28].

B. INSTRUCTIONS

The following steps describe how to use the Simulink® model of the SDR design for testing and synthesis. Initial versions of the *Test_control_testing* series of m-code files are designed to run MATLAB®/Simulink® tests simply by executing the file. This method was not used in the development of this thesis work because it prevents the operator from checking the intermediate progress of the test.

1. Examine the Simulink® Model

Open the desired Simulink® model. Check all m-code blocks and ensure that each m-file is included in the same directory as the model.

2. Conduct Incremental Execution of the Test File

Open the desired m-code test file. The test files are divided into multiple cells, each of which can be executed independently. The details of using cells are listed in the "Rapid Code Iteration Overview" section of [28]. The beginning of a cell is identified by a header comment in bold type. To evaluate an individual cell, move the text cursor within the cell. This highlights the cell. Select "Cell→ Evaluate Current Cell," or type "Ctrl+Enter" to evaluate this portion of m-code. Execute the first four cells in the file, ending with the cell labeled "Input Signal Generation." Check the MATLAB® workspace to ensure that all required input variables have been assigned. Once this is accomplished, the Simulink® model is ready for simulation and HDL generation.

The test file for models configured for three-chip design provides additional cells to reformat the output of each simulation so that it can be used as input for the next simulation. After the first four cells are executed, the simulation of the first model which contains the FFT IP is conducted. Once the simulation is complete, executing the cell labeled "Reformat FFT Output" adjusts the simulation output to be used as input for the second model which contains the Windowing Algorithm subsystem and Window

Analysis subsystem. Once the second simulation is complete, executing the cell labeled "Reformat Compression Output" adjusts the simulation output to be used as input for the third model, which contains the Temporary Memory subsystem, Format Output subsystem, and Downlink Control subsystem.

3. Synthesis

The process to create a FPGA configuration file from a System Generator Simulink® model is summarized in [3]. Specific details are provided in [10] and [11]. For this research, the design was compiled to the HDL netlist level by selecting this option and clicking "Generate" in the System Generator GUI. This creates a Xilinx ISE project, which can be opened using Xilinx ISE Project Navigator.

C. CHANGING PARAMETERS

The impact of adjusting the parameters *N* and *M* on the required depth of storage devices is discussed briefly in [3], which lists all storage devices and associated control algorithms. This information was clarified in Chapter III, which identified that not all storage devices are sensitive to changes in *N* and *M*. The new list of storage devices sensitive to changes in *N* and *M* is shown in Table 18.

Storage Device	**Write Control Module**	**Read Control Module**
FIFO (Time Wind)	*pwr_time*	*pwr_time*
Dual Port RAM (Freq Wind)	*we_time_win*	*re_freq_win*
Dual Port RAM (Real Data)	*we_temp_fft*	*re_temp*
Dual Port RAM (Imag Data)	*we_temp_fft*	*re_temp*
Dual Port RAM (Real Parity)	*we_temp_fft*	*re_temp*
Dual Port RAM (Imag Parity)	*we_temp_fft*	*re_temp*

Table 18. Storage Devices Sensitive to Changes in *N* and *M* [After 3].

In addition to the storage devices listed, the FFT error detection algorithm is sensitive to changes in *N* and *M*. As discussed in [15], changing *N* alters the timing performance of the FFTv4.1 IP block. The block would need to be retested to determine the latency between the first real signal input and the first FFT output point. This information can be used to adjust the delay blocks used to align the error detection signal with FFT output, as discussed in Chapter V. Comparison blocks used for timing in FFT error control algorithms are also dependent on the value of *N*. Changing the value of *M* requires a manual adjustment to the *ErrorFlagCtl* algorithm and the *out_hdr* algorithm because the length of the error code is equal to *M*.

APPENDIX B. ADDITIONAL APPLICATIONS

This section classified and is bound separately. Contact the Naval Postgraduate School Special Security Officer for access.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Atkins, A. Bass, et al., "An Operationally Responsive Space Architecture for 2025," Naval Postgraduate School, Monterey CA, NPS-SP-08-005, Jun. 2008.

[2] N. W. Bergmann and A. S. Dawood, "Adaptive Interfacing with Reconfigurable Computers," in *Proc. 6th Australasian Computer Systems Architecture Conference*, Queensland, Australia, 2001, pp. 11–18.

[3] D. A. Wright, "Field-Programmable Gate Array-Based Software Defined Radio," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2008.

[4] J. Phillips and B. Asiano, "Programmable Satellite Transceiver (PST) for Responsive Space," presented at the 6th Responsive Space Conference, Los Angeles, CA, Apr. 28–May 1, 2008, Paper RS6-2008-6004.

[5] J. Snodgrass, "Low-Power Fault Tolerance For Spacecraft FPGA-Based Numerical Computing," Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, 2006.

[6] M. A. Sullivan, "Reduced Precision Redundancy Applied to Arithmetic Operations in Field Programmable Gate Arrays for Satellite Control and Sensor Systems," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2008.

[7] J. Snodgrass, M. Surratt, H. H. Loomis, and A. A. Ross, "Single Event Upset (SEU) Testing of the Naval Postgraduate School Configurable Fault-Tolerant Processor (CFTP)," presented at the 9th Annual International MAPLD Conference, Washington, D.C., Sep. 26–28, 2006, Paper MAPLD 2006/147.

[8] M. A. Sullivan, H. H. Loomis, and A. A. Ross, "Employment of Reduced Precision Redundancy for Fault Tolerant FPGA Applications," in *Field Programmable Custom Computing Machines, 17th IEEE Symposium on*, Apr. 5-7, 2009, pp. 283–286.

[9] R. Cristi, *Modern Digital Signal Processing*. Pacific Grove, CA:Brooks/Cole Publishing, 2004, pp. 63–144.

[10] *System Generator for DSP, Release 10.1.1: Getting Started Guide*, Xilinx, Inc., April, 2008.

[11] *Xilinx ISE Design Suite 10.1 Software Manuals: Development System Reference Guide*, Xilinx, Inc., 2008.

[12] *Virtex™ 2.5V Field Programmable Gate Arrays*, Xilinx, Inc., Xilinx Product Specification DS003-1 (v2.5), April 2, 2001.

[13] *Virtex™-II Pro and Virtex™-II Pro X Platform FPGAs: Complete Data Sheet*, Xilinx, Inc., Xilinx Product Specification DS083 (v4.7), November 5, 2007.

[14] *Virtex™-4 Family Overview*, Xilinx, Inc., Xilinx® Product Specification DS112 (v3.0), September 28, 2007.

[15] *Fast Fourier Transform v4.1*, Xilinx, Inc., Xilinx® LogiCore Product Specification DS260, April 2, 2007.

[16] *Xilinx ISE Design Suite 10.1: System Generator*, Xilinx, Inc., 2008.

[17] *Xilinx ISE Design Suite 10.1 Software Manuals: Glossary*, Xilinx, Inc., 2008.

[18] *Xilinx ISE Design Suite 10.1 Software Manuals: Virtex™-4 Libraries Guide for HDL Designs*, Xilinx, Inc., 2008.

[19] *Libraries Guide*, Xilinx, Inc., ch. 6. Available: http://www.xilinx.com/itp/ (accessed December 6, 2009).

[20] *Xilinx ISE Design Suite 10.1: Project Navigator*, Xilinx, Inc., 2008.

[21] *Xilinx ISE Design Suite 10.1 Software Manuals: Virtex™-IIP Libraries Guide for HDL Designs*, Xilinx, Inc., 2008.

[22] *High-Performance 1024-Point Complex FFT/IFFT V1.0*, Xilinx, Inc., LogiCore Product Specification May 11, 2001.

[23] J. F. Wakerly, *Digital Design: Principles and Practices*. Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

[24] *System Generator for DSP, Release 10.1.1: Reference Guide*, Xilinx, Inc., April, 2008.

[25] R. C. Olsen, *Introduction to the Space Environment: PH 2514 Course Notes*, Naval Postgraduate School, January 2005, pp. 230–231.

[26] *Fast Fourier Transform v3.1*, Xilinx, Inc., Xilinx® LogiCore Product Specification DS260, November 11, 2004.

[27] *Fast Fourier Transform v3.2*, Xilinx, Inc., Xilinx® LogiCore Product Specification DS260, January 11, 2006.

[28] *MATLAB® 7: Desktop Tools and Development Environment*, The Mathworks, Inc., Natick, MA, September 2009, pp. 185–208.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Frank Kragh
   Naval Postgraduate School
   Monterey, California

4. Herschel Loomis
   Naval Postgraduate School
   Monterey, California

5. Alan Ross
   Naval Postgraduate School
   Monterey, California

6. Roberto Cristi
   Naval Postgraduate School
   Monterey, California

7. Alexander Julian
   Naval Postgraduate School
   Monterey, California

8. Donna Miller
   Naval Postgraduate School
   Monterey, California

9. Bieu Lu
   SPAWAR Systems Center
   San Diego, California

10. Jeremy Livingston
    NNWC GNOC Detachment
    Norfolk, Virginia